DOCUMENT RESUME

ED 052 485 24 CG 006 493

AUTHOR Brewster, David
TITLE Program Description-DASM: Direct Access Storage Management. Information System for Vocational Decisions.
INSTITUTION Harvard Univ., Cambridge, Mass. Graduate School of Education.
SPONS AGENCY Office of Education (DHEW), Washington, D.C.
REPORT NO PR-35
BUREAU NO BR-6-1819
PUB DATE May 70
GRANT OEG-1-6-061819-2240
NOTE 89p.

EDRS PRICE EDRS Price MF-$0.65 HC-$3.29
DESCRIPTORS Computers, Data Bases, Educational Technology, *Information Retrieval, Information Systems, *Man Machine Systems, *Occupational Guidance, *Program Descriptions, *Programing Languages

ABSTRACT

This paper explains DASM, a package of programs and routines for Direct Access Storage Management on 70/564 disc packs. Its basic unit of data is a record. DASM processes (retrieves and maintains) records by techniques which require the user to reference a record by its name (both record and file name). The steps through which a user can create a DASM file are examined. In addition the internal workings of DASM are clarified. (TL)

ED052485

INFORMATION SYSTEM FOR VOCATIONAL DECISIONS

Project Report No. 35

PROGRAM DESCRIPTION

DASM

Direct Access Storage Management

David Brewster

This paper was supported in part by Grant No. OEG-1-6-061819-2240 of the United States Office of Education under terms of the Vocational Education Act of 1963.

Graduate School of Education
Harvard University

May 1970

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
OFFICE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL OFFICE OF EDUCATION POSITION OR POLICY

CG 006 493

DASM

Direct Access Storage Management

DASM is a package of programs and routines for Direct Access Storage Management on 70/564 disc packs. The basic unit of data for DASM is a record. DASM processes (retrieves and maintains) records by techniques which require that the user reference a record by its name (both record and file name). The user does not specify nor know where the record is kept on the disc packs. These techniques for storage media management are based upon the concept of RASM, defined in Technical Report CA 6704-0422 April 1967, and an ISVD-RASM draft dated August 1967, both by Massachusetts Computer Associates.

Additionally, DASM has the following features:

1. Compatible with RCA-BTSS storage scheme and operation,
2. Consistent on-line and off-line processing,
3. Unique identification of records with equal name but different file names,
4. Automatic security checks,
5. Minimal core requirements,
6. Off-line disc pack purging and backup.

This paper will explain what a user must do to create a DASM file; and will show how DASM functions internally. There are four steps towards building and using a DASM file:

1. Condition a disc pack via BTSS initialization,
2. Build a system sub-media on the pack via DASM-10,
3. Manipulate data in the pack via DASM core resident routines (either on-line or off-line).
4. DASM off-line pack maintenance.

We will now examine each step.

CONDITIONING A DISC PACK VIA BTSS INITIALIZATION

Before DASM can use a pack, it must be conditioned through BTSS pack initialization (in SYSCRIBE). This routine is described in the SYSCRIBE documentation of BTSS. Basically, the routine builds a header

label on CYL-0, head-1, record-1: and writes one 2048 byte record of 55 (16) to all other tracks. The header label contains identification, an availability table on a track by track basis, and a defective track table. In addition, BTSS will reserve parts of the disc for swapping cylinders. DASM-10 will use the availability table as an internal check when it allocates DASM files.

To get a pack initialized, the procedure is:

1. Mount a clear pack on the system,
2. Mount a BTSS primary pack (already established),
3. Bring up a BTSS session from the primary pack,
4. When BTSS calls for Secondary Disc Assignments, type in the physical device number of the drive with the clean pack. Follow the device number with the letter I (e.g., 252I). This will cause the pack to be conditioned by BTSS. It will take about 6 minutes,
5. When the next console typeout occurs, your pack is conditioned (Note -- if it's an error typeout -- refer to BTSS operator's guide).

The pack is now ready to accept DASM sub-media and file allocations via DASM-10. Appendix A describes the disc pack after BTSS has initialized it.

DASM-10: BUILDING A SYSTEM SUB-MEDIA AND ALLOCATING FILE SPACE

DASM functions through the use of tables. Some tables are constructed by an off-line run (DASM-10) whenever files are allocated to the system. Other tables are generated and maintained dynamically to show the current status of DASM users during normal processing.

Once a pack is initialized by BTSS, DASM-10 can be run to condition a pack for DASM. The five functions of DASM-10 are:

1. Define a pack for DASM,
2. Define sub-media,
3. Define FILE,
4. Define USER AUTHORIZATION,
5. Recap Printout of the allocation on a disc.

Pack definition must be done first (and only once per pack). Sub-media, files, and user authorizations may be allocated as often as necessary.

However, in allocating user authorization to files, the files must be previously defined; and in assigning files to sub-media, the sub-media must have been previously assigned.

DASM-10 driven by parameter cards (see Appendix B) as an off-line run. These parameters are used to build a disc resident table referred to as a SYSTEM SUB-MEDIA. Each pack has a system sub-media reserved on cylinder 2, Tracks 0-9. A further breakdown of the system sub-media can be found in Appendix C.

During any session (off- or on-line) that uses DASM, the system's sub-media for each DASM pack is compressed into memory and used by the core resident DASM routines.

DASM CORE RESIDENT DATA MANIPULATION

Given a disc that has been initialized by BTSS and has file space allocated by DASM-10; the core-resident package is used to read, write, delete or rewrite records either on-line or off-line. (This package is assembled as DASM-20 in both cases.) To function, core resident routines require an area consisting of three separate tables. These tables are:

1. Secondary log-in areas (not required for off-line, but maintained for compatible coding -- a dummy user number = 1 is assumed off-line).
2. A simplified core resident version of each active pack's system sub-media.
3. An open file list.

A description of these tables is found in Appendix D. The tables must be established in core before any function involving a file or a record can occur. In both on-line and off-line, there is a routine to construct these tables. Off-line versions contain the coding at the beginning of the program. It is executed by requesting the open pack function.

In the on-line version, the first call to DASM automatically constructs the core resident tables. During this time, it is imperatus no after user requests DASM functions (about 1 minute). It is recommended that 1 person log on, then allow others too.

Whether on-line or off-line, the results are the same. The secondary log-in areas get initialized, the system's sub-media are abbreviated

and stored in core, and the area reserved for open files is linked together as an empty list structure. Control is then returned to the main program which will request the various data manipulation functions that are available. There are three classes of functions built into DASM. They are:

1. Those involving files, such as Open or Close,
2. Record manipulation,
3. Security and authorization requests.

Appendix E defines the various function codes in detail to request a function of DASM. The user builds a File Pointer Block (FPB) as described in Appendix F. He puts the address of the FPB on GR-1. Then, he transfers control to DASM by:

1. BAL 14, DSTART in the off-line version, or,
2. Gives an SVC 206 in the on-line version (he must be in state P-1).

DASM will return to the user at the next instruction. The proper return code will be set in the FPB. Appendix H specifies the response codes that are stored in the first half-word of the FPB.

The next section involves the mechanics and the theory of on-line DASM as it is currently implemented. Those relying on on-line functions should be familiar with it.

UNIQUE CHARACTERISTICS OF ON-LINE DASM

On-line DASM depends on BTSS for its I/O functions, some of its storage allocation, and to receive key information regarding the active user. Additionally, since BTSS is a time sharing system and because DASM functions are quite lengthy in time required to complete, DASM must be able to relinquish control back to BTSS whenever it is waiting for I/O. This requires DASM to be re-enterable.

Workfiles are a feature in BTSS that are used by FORTRAN. Since ISVD has taken FORTRAN from BTSS, they are not used. In each user's log-in area, there is room for 4 SIX BYTE EDA'S for workfiles. (EDA'S are used by BTSS for I/O Requests to the disc.) DASM utilizes workfile's EDA'S for disc requests.

DASM on-line functions execute in P-2. The core resident tables must lie in location lower than 65535(10). This is because of half-word

addressing techniques. The program calling on DASM must be running in state P-1. This is because MAINSC sets registers 1 and 15 in state P-2 when SVC'S come from P-1. The program that calls on DASM gives an SVC 206 (formerly used for workfiles).

DASM has one entry point (DASMMAIN). DASM gets control here on the initial SVC and whenever it re-enters after it previously relinquished control to await I/O completions. A re-entry address is stored in each user's secondary log-in area. When control is received at DASMMAIN, a check is made of the re-entry address. If it is 0000, then DASM assumes it is the first time through for this user. Otherwise, it knows the address at RE-ENTRY is where it left off on the last time slice. Before returning control to the caller, DASM resets RE-ENTRY to 0000.

When requesting Disc I/O, DASM builds on EDA at the first three bytes of the workfiles EDA. At present DASM can maintain only 1 EDA and further expansion would complicate matters greatly.

Whenever Disc I/O is needed, DASM sets the IFR bits that reflect clock and SVC interrupts in the user's primary log-in area. DASM then fires off an SVC RETURN. When the requested data is in core and assigned a half page, BTSS MAINSC will return to DASM at DSTART. DASM uses the re-entry address as the secondary log area to get back in place.

When all DASM processing is complete, DASM gives an SVC RETURN (without setting IFR clock and SVC bits). The user of DASM is responsible for releasing the half page with DASM data on it. This is done by altering the workfile EDA for the active user.

At present, on-line DASM allows only 1 user to be actively reading or writing. Simultaneous requests are locked out automatically by throwing extra users into a "busy" loop.

This was necessitated by concurrent users wishing to update data in the same track on a disc simultaneously.

Users #1 and 3 or 04, in consecutive time slices, wish to update records. The effect would be, both get the original value of the track on their read, the first user would write or lock with his corrections, the second user would then write both the original track with his corrections. This would cancel out the first user's update.

During an on-line session, it is possible for the system to"bomb" amidst a DASM operation. Results are unpredictable, and cause great grief.

Typical symptoms are having an index entry partially modified, or not having the data portions stored. To remedy this situation, the user should obtain a "SLUMP" of all memory. The systems analyst can then inspect the secondary log-in areas (DASM'S) to see how far each user had gone before the alert. Since only 1 user could be a writer (this is where the trouble occurs), it is easy to deck his re-entry word and determine his exact progress. In some instances, channel registers can show how many data bytes have been transferred. The cylinder and track number are also stored in secondary log areas. The recommended procedure would be to Dump the cylinder using RAEDIT and find the status on disc. To assert "fixing" these runs, a routine called DSMHELP has been written. This allows the analyst to clean off undesirable records from a specified cylinder.

During the active sessions, records are frequently rewritten. Re-write logic does not drop the old record, but merely flags the proper index entry, then rewrites the modified record at the end of the used area within the cylinder. This causes a considerable amount of wasted space and eventually leads to exhaustive overflow conditions on the pads. Two routines, DASM-30 and DASM-40 has been written to maintain packs and clean off deleted records off-line.

DASM-30 will write all active records from a pack onto a reel of tape. As records are written to tape unblocked, a line is printed to reflect each record's status, good (G) or deleted (D). Only "G" records go on the tape. Each record is preceded by a 66 byte Leader, specifies file and record name plus length of all names. D-30 writes active records from the entire pack.

The data carried in the 66 byte header of a D-30 tape record is used by D-40 to reload "active" records onto a clean disc pack (cleared by BTSS and D-10 as specified earlier).

Optimal placement could be achieved by running a Sort between D-30 and D-40. The criteria could be record size, priority or some yet undefined criteria.

DASM-40 reads records from a tape and "writes" them to disc, LO assumes the disc has been cleared off and its system's sub-media reconstructed via D-10. It utilizes the tables of the system's sub-media to recalculate the record's location. This feature allows us to re-allocate areas over the disc with minimum core.

DASM APPENDIX A

BTSS PACK ALLOCATION

CYLINDER-0, HEAD-0, R1: BOOTSTRAP-not referred to by DASM
CYLINDER-0, HEAD-1, R1: 850 byte record as:

18 Byte Header = TIME SHARING DISC
20 Bytes = data from console
406 byte = Track Acceptability. 2 Bytes per cylinder showing good tracks as 0, normally as 003F(16)
406 byte = Track Availability. A copy of Track Availability Matrix but with "1" bits reflecting allocated areas in addition to unacceptable tracks. BTSS reserves swapping cylinders here. DASM adds in "1" bits to any areas it will reserve including the System Sub-Media.

Cylinder 0, Head 2 then Cylinder 202, Head 9 are set to contain 1 record each, 2048 bytes long, and contain all 55(16).

DASM APPENDIX B

PROGRAM SPECS FOR DASM-1C

Purpose: The purpose of DASM-OFF-LINE INITIALIZATION is to set basic DASM control facts on a disc pack. It can be used to:

A. define a pack
B. enter sub-media range
C. enter files into the catalogs
D. enable users to have special privileges to a file
E. print a recap of a pack's contents

Parameters, specifying what is to be done, are supplied through punch cards.

Conditions: The pack should previously be defined by BTSS - SYSCRIBE.

Devices: A) Disc, Card Reader, Printer

N.B.: Various types of names are called for as parameters. Most have a maximum length of eight bytes. All should be entered beginning with a non-blank character in the left-most column.

Parameter: Define Pack Parameter

Col	Field	Purpose
1	'P'	specify pack parameter
2	Δ	
3-10	pack name	uniquely identify pack
11	Δ	
[12-24	'BTSS HAS CABS'	allows option to skip BTSS files
25]	Δ	
[26-28/29-31/etc.	XX,	cabinets, reserved for BTSS are string

This parameter will cause the disc pack on the console assigned drive to be conditioned for DASM: (The System Sub-Media will be defined at Cyl. 2 by this run.) It assumes that BTSS has previously initialized the pack by writing BTSS labels, assigning alternates and reserving any cabinets it may require. Should this pack require further areas allocated, the optional values starting in column 12 are entered. All DASM buckets are conditioned with a standard DASM leader. Example: to define a pack as 'TEST' and reserve 10, 11, 12, 13, 14, and 15, the card would appear as:

PΔTESTΔΔΔΔΔBTSSΔHASΔCABSΔ10, 11, 12, 13, 14, 15.

[ ] defined optional parameter

DASM OFF-LINE INITIALIZATION

Parameter: Sub-Media Definition

Col	Field	Purpose
1	'S'	specify sub-media parameter
2	Δ	
3-10	Sub-media name	
11	Δ	
12-19	pack name	
20	Δ	
21-23	starting cylinder	define extent
24	Δ	
25-27	ending cylinder	" "
28	Δ	
29	type sub-media	s=system, u=user w=work

Function: The sub-media range defined here is checked to see if it overlaps any BTSS file. If so, the run terminates with appropriate messages. Otherwise the sub-media is registered in the system sub-media table.

DASM OFF-LINE INITIALIZATION

Parameter: File Definition

Col	Field	Purpose
1	'F'	specify file
2	Δ	
3-10	file name	
11	Δ	
12-19	sub-media name	
20	Δ	
21-24	owner's nickname	
25	Δ	
26	type file	w=work, u=user s=system
27	Δ	
28-29	FILE MODE CODE	not implemented

Function: The file is listed in the system sub-media according to the parameter card.

----------NOT IMPLEMENTED----------

FILE MODES and OPTIONS

NO = Normal	RO = Read Only	WO = Write Only
OPA = Option A,	LR = Limited RO,	PR = Public RO
OPB = Option B,	LW = Limited WO,	PW = Public WO

These may exist in combinations which do not violate the following rules:

1.) NO = NOT.RO.AND...NOT.WO
2.) LR = RO.AND...NOT.OPA
3.) LW = WO.AND...NOT.OPB
4.) PR = RO.AND.OPA
5.) PW = WO.AND.OPB

FILE MODE CODE

File mode is indicated by a combination of the following codes:

mode/option	OPA	OPB	RO	WO
code	1	2	4	8

Combinations are formed per the rules stated under FILE MODES and OPTIONS; with .NOT.RO indicating that the RO code is omitted.

1.) NO - code is 0, 1, 2, or 3
2.) LR - code is 4, 6, 12, or 14
3.) PR - code is 5, 7, 13, or 15
4.) LW - code is 8, 9, 12, or 13
5.) PW - code is 10, 11, 14, or 15

DASM OFF-LINE INITIALIZATION

Parameter: User access to file

Col	Field	Purpose
1	U	defines user parameter
2	Δ	
3-6	user's nickname	user's nickname
7	Δ	
8-15	file name	
16	Δ	
17	permit code	

Function: This allows some users different access than all users to a given file. The appropriate entry is made in System's Sub-Media.

DASM OFF-LINE INITIALIZATION

Parameter: End of pack initialization

Col	Field	Purpose
1	E	defines end of initialization session
2	Δ	
[3	OPTION='R']	

Function: A summary report is proposed. If the option selected is R, then a listing is made of the System Sub-Media.

DASM APPENDIX C

Systems Sub-Media Structure

Track(s)	Elements and Their Length
0	Pack name (8)
0	Pack availability (800)
0	No. of sub-media this pack (2)
0	" " files " " (2)
0	" " listed user names this pack (2)
0,1	String of sub-media entries (16)
2-5	String of file entries (26)
6-9	String of listed user names entries (14)

Sub-Media Entry (16 Bytes)

Bytes	Field	Use
0-7	S.M. name	Identification
8-9	First cylinder allocated	Base address
10-11	No. cylinders allocated this S.M.	Used in hashing
12-13	# files this sub-media	
14-15	Sub-Media flags and type	

File Entry (26 Bytes)

Bytes	Field	Use
0-7	File name	
8-9	Keyword	Used to identify records uniquely
10	Type file	Work/User/System
11	Standard file protect byte	Generated for most user authority
12-15	Owner's nickname	
16-17	# users having READ ACCESS PERMITs	
18-19	" " " WRITE ACCESS PERMITs	
20-21	" " " REWRITE ACCESS PERMITs	
22-23	" " " FULL ACCESS PERMITs	
24-25	Sub-media this file belongs to (as a sequential number)	

There is room for 78 integral records to a track. An entry does <u>not</u>.

Listed User Names (10 Bytes)

Bytes	Field	Use
0-3	User's nickname	
4-5	File's keyword	Identifies which file
6-7	File's sub-media number	" " sub-media
8-9	Status and authorization of user	

DASM APPENDIX D

CORE RESIDENT TABLES

Structure of Core Resident Table. Total area is bytes.

A. Constant area of 728 bytes
B. A variable length list for each on-line pack:
   It's length is a function of the mix of sub-media or file composition.
C. A threaded list for open file entries

Table 1 - Constant Core Resident Table

A. 10 Secondary Log-In Areas, each 72 bytes long

1. Log-In No. (2)
2. Current Status (2)
3. No. open files (2)
4. Current file pointer (2)
5. Record Name Length (2)
6. Record Name (32)
7. Used to Store Re-entrant information (18)
8. Re-entry Jumppoint (4)
9. GR 15 from State P1 (4)
10. FPB Pointer (GR 1 from P-1) (4)

B. Address link to available slot in open file list (2)
C. Length of table read in at SYSCRIBE (2)
D. Open file list header (2)
E. No. packs on-line (2)

Table 2 - For each pack on-line a table is built as:

A. Pack Name (8)
B. No. of S.M. this pack (2)
C. Physical Location of this Pack (2)

For each sub-media, the entry is as follows:

A. Sub-media name (8)
B. First track address (2) contains track and drive numbers also
C. No. tracks (2)
D. S.M. flags (2)
E. No. files this S.M. (2)

For each file (or 1 per group as in students) an entry exists as:

A. File name (8)
B. File Switch and Status (2)
C. Owner's nickname (4)
D. Keyword (2)

Table 3

As files are opened and placed in use, entries are placed on a threaded list, the open file list. Each entry on the open file list is constructed as:

A. Link to next entry (2)
B. File name (8)
C. Keyword (2)
D. S.M. link (points to S.M. entry in current table) (2)
E. Current File status
F. Terminal authority (16) one byte per possible user.

(FUNCTION CODE) 16	FUNCTION
Ø2	OPEN file (OPEN) (wt/wtout explicit authorization request)
Ø3	CLOSE file (not implemented)
Ø5	SETMODE (not implemented) (per two masks; one for sets, one for resets)
Ø6	LOCKOUT (not implemented) (locks out other users)
Ø7	RSLCKOT (not implemented) (reset lockout)
ØB	FETCHR (READY) (retrieve record)
ØC	INSERTR (WRITE) (add only new record)
ØD	REPLACR-(REWRITE) (replace existing record only)
ØE	STORER - (WRITE OR REWRITE) (store record; replacing existing record if nec.)
ØF	DELFETR (DELETE) (delete record from file, preceded by FETCHR)
1Ø	DELETER (DELETE) (delete record from file, without preceding FETCHR of the record itself)
11	DELETEF (not implemented) (delete all records of file)
16	ROON (not implemented) (set file mode RO on)
17	ROOFF (not implemented) (set file mode RO off)
1A	WOON (not implemented) (set file mode WO on)
1B	WOOFF (not implemented) (set file mode WO off)

*Names in parenthesis refer to name of relevant routines in DASM code.

DASM APPENDIX F

FPB FORMAT

The format of an FPB is:

```
        DS ØF
        DS H        16-bit error flags (see Appendix H)
        DS C        Function code (current or previous)
                    (see Appendix E)
        DS C        Character count of filename
                    (True binary count:  1 to 8)
        DS CL8      Filename left-adjusted and padded
                    with null bytes
        DS C        Character count of record name
        DS CL32     Record Name
        DS 18C      Not used by DASM, required by HASM compatibility
        DS C        EDA number, not used in current versions
        DS 26       Not used
        DS H        Record Length
        DS F        Address of first date byte
        DS 2F       Used by HASM
```

DASM APPENDIX G

DASM BUCKET FORMAT AND CONVENTIONS

1. Indices start at Head 9 and go up.

2. Data starts at Head 0 and go down.

3. All index tracks are preceded by a 16 byte header; its format is:

0-3	Cylinder title	'DASM' (only at H-9)
4-7	Location	Cylinder # (only at H-9) in packed decimal
8-9	No. entries this track	Binary: # index entries
10-11	No. words used on index	Binary: # words used on index
12-13	Not currently used	
14-15	Next address If 0, no more index tracks/cylinder If FF, go to meet cylinder Otherwise go to next usable index tracks in this cylinder	

4. Each Index entry is word aligned. Its format is:

0-1	Length of Record Name	Includes Keyword (at 2)
2-3	Length of data	
4	Flag to purge, 'X'09'	
5	Not currently used	
6-7	Priority	Values 0-15
8-9	Keyword	
10-?	Record name	

5. Data entries are stacked sequentially on a byte aligned basis. There are no data entry delimiters. However, data records are always stored entirely within a track. If, when tracks are created, there isn't room for a data entry, it is stored on the next track, also true of index tracks.

1. Logic of Hashing Scheme.

   The Hashing Scheme accepts any string of bytes as an input argument. (There are no restrictions in the scheme itself, however DASM imposed a length of 255 as a maximum string length.)

2. The result of the HASH will produce a 32 bit value in a general register. The user is responsible for dividing the hash result by the number of addressable cells (buckets) that he has allocated for his file.

3. The hash result is calculated as follows:

   Initially, the 32 bit 'sum' is set to 0000.

   The scheme then scans the string from left to right. It picks up 4 bytes at a time (except possibly when it takes the end of the string), and 'EXCLUSIVE OR's it to the register that contains the sum.

   Example of HASH scheme:

	scan cycle		sum
	initial		0000
'SAMPLEØFNAME'	1	SAMP	SAMP
	2	LEØF	LEØF 'X' SAMP
	3	NAME	LEØF 'X' SAMP 'X' NAME

   where 'X' means EXCLUSIVE OR

   There is, however, no guarantee the string argument will be a multiple of 4 bytes. When the string is not a multiple of 4 bytes, the hashing scheme will 'pad' on the last scan.

   Padding fills out the last scan segment to make it 4 bytes long. It uses the first byte of the string as the pad value.

   Example of HASH when padding is done:

string = 'ABCDEF'	scan segment	scan #	sum
		initial	0000
	ABCD	1	ABCD
'padding'	EFAA	2	ABCD 'X' EFAA

4. The string argument used as input should be followed by a period to stop the coded hash algorithm. In DASM, I use the length of name to compute where the period is stored.

5. Caution! When you get the HASH SUM, it is a 32 bit logical sum. Use a 'Load Positive Register' command to make the sum positive, otherwise a negative value could louse things up.

Example of HASH

```
string 'DASM 138'
where in HEX it's  'C4 C1 E2 D4 F1 F3 F8'
in binary it's     '11000100-11000001-11100010-11010100-11110001-11110011-11111000'

Segment        Scan                                              Sum

               initial                        0000 000 000000000 0000 0000 00000000

'DASM'           1          'X'

11000100 1100 0001 1110 0010 1101 0100

                                          1100 0100 11000001 11100010 1101 01000


'138'            2
padded into            'X'
138D

11110001 11110011 11111000 11000100          1100 0100 11000001 11100010 11010100
                                             1111 0001 11110011 11111000 11000100

                 SUM =                         00110101100110010 0001101000010000
```

This number is then divided by the number of buckets, and the remainder is the bucket # where the record should be.

DASM APPENDIX I

On-Line DASM (DASMSVCE)

	DASMSVCE Reg. 3 is set up as temporary Base	DASM always gets control from BTSS at this point via SVC 206. This is true even if control is returning after A Read or write.
START	Reg. 14 set to initial path	
	All registers are saved for BTSS (o-15 in P_2)	
	Registers 10-12 see set up as Base registers. Register 13 points to table area.	Set up Base Registers (all are not required) 10+13 are only ones required.
	DASMBSX Set Gr_9 - '79872', Location	
DASMBSX	of first BTSS Log is Set Gr 8 = 1	BTSS puts address of Login area in Gr 15. I try to convert it to value 0-10. The algorithm scan all possible BTSS login addresses and makes matching value. In ISVD there can only be 10 users, with 11 possible login areas in use.
	Does $Gr_9 = Gr_{15}$ Yes	
	Nø	
	DSMPLGNX	
	Add 1 to Gr_8 Add 366 (length log-in) to Gr_9	Here if I can find login #.
	Is Gr_9 at place	
Nø	for 11th Login? Yes	If this occurs, BTSS has sent me an invalid Login #.
	Systems crash - cause bomb and dump.	

DSMPLGNØ	Put Gr_8---Gr_{15} then reduce it by 1.		Set 15 = value 0-15 to represent DASM's secondary Login number.
DSMSWTCH	Load Reg. 3 with value at DASMJUMP Branch to (Gr_3)		DASMJUMP is loaded as address of 'DSOPNPCK'. This causes control to go into pack open routines automatically when first call is made to DASM. The end of the pack opening routines, sets 'DASMJUMP' into regular main line logic.
DSOPNPCK	Have I used the open pack routine at all during this session	NØ	This is determined by checking 'DSLGIN1', which loads as a negative value. If first time I set it to value of register 15.
	Yes		
	DSTR1ST		
	Is it the same user I started to open packs for?	NØ	In here I check to see if the user I'm processing now is the same one I started for, if not I let him wait until I'm done.
	Set Gr_{14} to value saved in his Sec. Log is free. This is user's re-entry point.	DSMLØTGØ	
	Go to (14)		

DSTR1ST	Store Gr_{15}---DSCLGIN then set Gr_2 = start of secondary log areas.	Initialize everything to clean off.
DSTSCLG	clean of 10 secondary Login areas (each 72 long) to X 'oo'	Clean secondary Logins and initialize list headers
Dclean	clean off all remaining table area to x 'oo's	
	Set Register 6 to start of remaining table area (After Logins and Headers)	
	Set EDA's up to read cabinet 0 cylinder? For logical drive 0 save Gr_6 as next free area	Set up for DISC read in the System Sub Media
NXTPACK	Restore Reg 15 as it was from BTSS and put it in My Sec. Login area.	This step enables DSIORD to function properly.
	Set Head = X.^{oo} for Read	
	Go to DSIORD subroutine	

Go to
Routine
DSMSET 3'
to set Gr_3 =
location of DATA

Restore 6 =
start of free area

is it a
DASM pack? N∅

check if system sub media area is set to x'5555'. If so, it's not a DASM pack.

Yes

DNWTHPCK

Add 1 to on live packs
(DOLPCKS)
Put pack name---Table
SMloop Get #Sm, #P6S on
pack. Put logical
drive #---Sec longin area

If DASM pack, set up contents of SSM in core resident table.

Go into loop to build
all sub-media
entries: Assume all
SM entries on 1 track.
Save '6' at DSMWIA
after each.

As I build SM entries, I leave room for the file entries.

Gr 6 always points to next free space.

Read in File
NHSMPCK track.

Set Reg 3 to
Buffer Address

Put file entries
into SM slots
Lookatfl

This routine puts files into already built slots in sub media. It uses sequential counts in Gr_0 to match them.

Yes

Any files I
can't match up

n∅

Bomb

DNWTHPCK

Was that logical Pack '3'

Yes

After matching up files I am through initializing that pack, I now check to see if other packs are on system.

Nø

QUITINIT

Add 1 to Logical Pack number and store it at Login area.

Branch back to NXTPCK

QUITINIT

Come here after doing all packs. (Max. of 4) in system.

Build available list of entries in remaining area for open files

build open file list.

All done initializing

Set 'DASMJUMP' to address of DASMRPOL

This causes all calls to DASM to go on regular path. 'DASMJUMP' initially sends you to initialize routine.

DASMRPOL

DASMREGL

This is where all calls come other than the first, which causes packs to open, then it comes here too.

Check if it's the first time through for this call.

Yes / Nø

A word in the Sec Login area is initialized to 'o'. If I am doing something for a user I store Gr_{14} at that word. (14= Line register). Whenever I return to BTSS, I store 14. When I'm done with a users request, I 'o' out the word.

Save Registers 1 and 15 in his Login area:
Set Gr_{14} to 'DSMTLUCC-4' which is start up logic

Reset Gr_1 for him, and set Gr_{14} = his re-entry point

Get FILENAME→'DSMFLNM'

Set up parameters (FPB,name) from FPB into fixed area.

Restore all variables from Login---fixed area

Many variables must be stored for re-entrant problems - DSMDL, DSMDT, DSMIT, DSMTRK, DSMWHRAT, DSMDRPAD, DATAR, DSMCAB, ADDT, DSMCYL. This is done for re-entrant reasons.

Set Gr^2 = start of name.

Go to value in Gr 14

Go to wherever you were for this user.

	Do a table look up against function code		I come here only first time for each new call.
DSMTLUCC	in FPB. Find my internal code and location to jump to.		I search permissable function codes and if not there, I return with invalid function. Otherwise I pick up an internal code (established prior to FPB's) and a jump location. This is found in DSMCCTBL
	Was I able to match up the op code requested?	Nø	
	Yes		DSMLNVFN
		Invalid function- set error code in FPB	
DSMINVFN			Invalid function routine
	I've matched up code- set up jump point make data lengths even.	Go to DSM99	This is ideal place to get data lengths even, in case HASM hasn't.
	Is the		
SENSE	system busy reading/writing for someone else.	Yes	At this point I check 2 switch called 'WRSWTCH'. It's purpose is to lock out other users when I'm reading/writing for any user.
	Nø		
		Set register 14= address of sense Routine	The 'DSMFIND' routine sets the switch as it does all I/C.
		Go to DSMLETGO	
LETTHRU			
	Jump to routine to perform function (in Reg 14)		
	All return below		Leave Mainline DASM + Go to proper subroutine. This is found in function/jump table.

Reset the busy switch: turn it off.

DSMCMBCK — Set re-entry point for user = o.

I always come back here after finishing a function.

Was it a read — Nø

important to reset re-entry point for a user to o as I'm all done with him.

Yes

Store the data length is FPB

BUG1969

Check for any errors — Nø

I carry an internal error flag at DSCRSTW for each user. I check this and convert it to

Yes

DSM1X
DSM2X — Set proper Bit setting in FPB

DSM99 — Restore all register for BTSS (P_2 state) and leave by SVC 171

This tells BTSS I'm all done with a function for a user.

Subroutine to open file (or determine if open)

called by BAL 14, DSMOPFFN.

This is DASM routine to open a file. IB called each time a read/write function is requested. It links thru 14.

DSMOPFFN — Set pointers to first entry on open file list

I first check to see if the file is already open.

DSMOPSCN — Are there any more entries open in open file list — Nø

Yes — DSMOPFFL

Is the file I want open? — Nø

Yes

Step to next entry on open file list.

Leave Reg 1 pointing to open file

If I find file already open

Return via 14

DSMOPFFL — Calculate # of packs on line from 'DOLPCKS'

I come here if the file I want isn't already open.

Have I exhausted all packs — Yes

Nø

For each on line pack, I will go thru and search files in a serial fashion.

DSMTDL

DSMØPL3 — Figure out 3 sub media this pack

For each sub-media, I go into file scan loop

DSMØPL2 — Get # files this sub-media

Check file name remembering which sub media I'm in.

Is this the file I want? — Yes

Nø

DSMOPFFL

Step back thru DSMØPL3, DSMØPL2 for all packs

Loop mechanism for all packs.

Yes — Are any packs left?

Nø

Set error in FPB'file not found'

Couldn't find file - set error and return to TRAC

DSMCMBCK

DSMOPYSI

Get next available open file slot in 'DAVL' list.

I've found file, put it in open file list.

Is 'DAVL' empty — Yes

Have I exhausted my storage.

Nø

DSMIDL

DSMIDL causes a bomb.

Put this file in open file list

- - - - - - - - - -

step 'DAVL' to next cell

'DAVL' is list reader for threaded list of cells

update statistics on open file

Grl is set to address of entry.

This is subroutine to open file
I leave via 14.

Leave via 14

Hashing Subroutine
Linked via Register 14

DSMHSH — 0's ---- Logical Value — initialize my hash sum of record name

DSMHSH LP — Pick up max of 14 char in name at a time — I go across name, getting 4 characters at a time. A period is set at the end of the name. If I picked up < 4 char, (because name ended), I pad out 4 characters with value I have in logical sum.

Were there 4 char that I just picked up. — Nø

Yes

DSMHSHPD

DSMHSHXC — Exclusive or the 4 char. to to logical sum

Was that the end of the record name. — Nø — I index across the name

Yes

DSMHSHLP

Return via 14 — The calling routine gets a 32 bit logical sum in 'DSMHSHWD'. He is responsible to process it into an address.

DSMHSHPD — pad out enough Bytes (1-3) to get 4 Byte value — To pad, I use the left most Byte in the 4 char. area I'm padding.

DSMHSHXC

DASM Main Line Logic

			'DSMFIND' is used by all functions (except a few minor ones)
DSMFIND	Save links, set 'WRSWTCH' to not found.		Initialize switch for rewrite, store links.
	Use the open file routine		Gr1 will point to open file entry upon return.
	Was the command an 'open file'	Yes	This is all I do for an 'open file'

DSMFEXIT

			If not open, I must transfer data in or out of cpu. This involves getting index piece and data piece
DSMFAUTH	Save pointer at this user's status.		
	Set up record name via specs for HASH routine		
DSMSVRN			Put a period after the last character, after insuring it's even length record name
ØK91168			
	Use the HASH routine		

	Divide HASH sum by # cylinders in the proper sub-media	
	Use the remainder of division, added to base of SM to get cylinder	This gives me an initial point (home) to look for data.
	Set switch 'DRPAD' and save 'Home' address at DSEDA - 10	Save initial 'Home' point for overflow test to see if SM is saturated. Set Drop/Add switch off.
DSMØVRTN		I come back to here when I'm in cylinder overflow routine.
	Remember this cyl # at 'DSMSHRAT'	
	Use the 'CABCYL' routine	This converts my logical cylinder NO. (003-202) into BBS cabinet (0-25) and cylinder (0-7) format.
	Initialize all sums of data lengths to zeros.	
	Set track = 9	This is index track.

Read address
set in
DSMTKRed CAB/CYC/TK

DASM gives up control
in Read routine

When I get control
again (at DASMMAIN)
I use re-entry jump
in users Login area to
come back here.

GIVE UP CONTROL

I must calculate where
data is by using
'Set 3'.

Use
'DSMSet3'
DSMRNTRT Save "3"
in 'DSMSET 3'

I use $G1_3$ as
pointer to current
piece of data I'm on.

Is this
a new track Nø

I look at index track
header (ITH). If it's
zero, then this is a new
track.

Yes

Make an
'ITH' as this
will become
index track

DSMFY2

Index entries start 16
Bytes in, past the 'ITH'

Set Register 3
to start of
index routines

Are there any entries left to look at.

N∅

Yes

DSMCHKNI

DSMFANIP

I go through several checks to see if entry I'm at matches the record name in FPB.

Do the record name lengths have same lengths

N∅

Yes

DSMSTPNI

Do record names agree?

N∅

This is done under 'execute' control

Yes

DSMSTPNI

Check keywords

N∅

Yes

DSMSTPNI

DSMSVP4

Am I Reading

Yes

N∅

I found entry that matches!

DSMFEX1R

On reads, Rewrites and Priority functions this is good.
On writes, it's an error this is main jump off point from common logic into specific functions.

Was it
rewrite Yes

Nø

DSMREWRT

Was it
a Yes
delete

Nø

DSMDELET

Was it
set priority Yes

Nø

Set
Prior

Was it
get priority Yes

Get
Prior

This is end of commands
where I expect a match.

Return error

It's a write, so I
send back error bit
saying it's already
there

DSMFEXIT

Get Prior	Put priority in FPB, from index entry	Return this records priority in the FPB and exit
	DSMCMBCK	
Set	Put priority code in index track	Set the priority at the FPB into the index entry
	'Write' put this track back.	Then write the index record back to disc.

GIVE UP CONTROL

	DSMCMBCK	return to final exit

RMVSR	Search open files for - FILE NAME	Remove user's file from open file list
	Zero the slot and return it to open file. Available List	I return open file entry that I just emptied.
	DSM99	

DSMSTPNI

Add data length this piece to 'SUMDL'

This is where I account for piece I just looked at.

Would it have fit on track

Yes

I adjust the running sum of Data lengths,

Nø

adjusting for integral records on a data track. (Data is not in core yet.)

Add 1 to DSMDT, this is DATA track on cylinder

Then I step to next index entry position on index track

Put DATA length last entry as 'SUMDL'

Add length of record name to constant '10' to get next entry

Reduce piece count by 1

I check to see if any more entries on the track. The piece count initially is found in the 'ITH'

Any more pieces on track?

Yes

Nø

DSMFANIP

DSMCHKNI

I've looked at entire index track. If there are more index tracks in the cylinder, I go to next one and continue search,

DSMCHKNI

Does chain address specify cyl overflow — Yes

Nø (x'255')

otherwise I check for cylinder overflow.

DSMGTØVR

Is there another index track — Yes

Nø

DSMGTNXT

I've exhausted search.

Was I trying to 'write' — Yes

Nø

On a 'write' this is expected.

DSMWRTIT

Set error bits in 'FPB'
Record not found

DSMFEXIT

DSMWRTIT	Was I trying a delete	Yes	This was an after thought.
		DSBUG349	This is best spot to check for delete and not found
	Remember where you are on index track for later adjusting		I was writing
			Check to see where I fit the data and the index entry.
DSMWLITF	Will this new entry fit on current track	Nø	
		DSMNXT	
	Make an index entry (make even length)		Put index entry on current index track
DSMFY3			
DSMSTRDT	Will data fit on current track?	Yes	Check to see where data goes.
		DSMPUTDT	If it does on current or next data track, I do so at DSMPUTDT.
	Can I fit data on next data track?	Yes	
			Otherwise, I retract index entry and go into overflow logic.
Nø			
		DSMPUTDT	

DSCLEAN	clean index entry I started set overflow = 'FF'	DATA won't fit in this cylinder
	Go to a 'diagnostic' routine to check format	(this is optional)
	write index track back	put index track back with overflow from cylinder set.

GIVE UP CONTROL----------

DSMGTØUR

DSMPUTDT	Finish index entry and add '1' to piece count in 'ITH'	When I find a cylinder to hold both the index entry and data, I come here
	Diagnose	(optional to check validity of index track)
	Write the index track back	

GIVE UP CONTROL----------

Build address
DSMRDTRK to read
data track.

Read data track in

Read
data track

GIVE UP CONTROL----------

There are checks in here, but
they are no longer necessary.

Move DATA
DSMUSEDT into track by
Loop.

Write
DATA
track

GIVE UP CONTROL----------

There is redundant code in
here, when I return. It
DSMFEXIT is unnecessary.

	SUBROUTINE		
DIAGNOS	Check all counts and Lengths.		This routine checks the validity of the index track before I write it back.
	Any errors?	Yes	It is optional!
		SVCBOMB	It's fatal if I find error.
	Return via 2		

DSMNXT			
	Is there another index track?	Yes	I come here to try for another index track in the same cylinder.
		DSMMKCHN	
	READ current index track		Must make overflow
DSTØK			

GIVE UP CONTROL----------

Set Gr_3 = input area		Go into search for first available cylinder. This is where you store overflow. It is done on a linear search method.

DSMTØVF

Are you at end of sub media (last track)

Nø

This is overflow routine

Yes

cycle around to first track in sub media = Reg 2

increment by 1 from current track to next = Reg 2

Go to next track or wrap around

Is track address in 2 = original Hash

Nø

have I exhausted SM?

Yes

DSØVRTN

Set sub media fill bit in FPB

Sub Media fill notication
or
Can't find on READ.
Depends if Read or Write.

DSMFEXIT

DSMMKCHN

Make CAB/CYL overflow pointer for this index track (put in 'ITH')

I get here if I can get another index track in the cylinder, without cylinder overflow.

Write this index track back

This routine is used when I create an overflow index track in the same cylinder.

GIVE UP CONTROL----------

Add 1 to current track I'm at

return and scan next track

DSTKØK

READING AND FOUND MATCH.

DSMFEX1R

Use DATA Length
to determine if
it's on current
DATA TRACK ('DSMDT')
or next one.

Calculate where
data track will be

(I've already determined
I can't overflow)

READ
DATA
TRACK.

GIVE UP CONTROL----------

Set Reg 3 =
to start of
record

This address in 3, will
be stored in Users
FPB at Record Pointer.

DSMFEXIT

RELHP	Calculate from Record Pointer in FPB (shift off Low order 11 bits)	Find the half page
	Set Bits in EDA to Let HP go from core	Tell BTSS where it is.
	B DSMLCTGØ	Give up control.

DSMGTNXT	Calculates next index track = current track plus 1.	This routine just figures out, the next index overflow track is 1 above where you now are in the same cylinder
	DSMTRKRD	
DSMCABCYL	Convert cylinder # (000-202) = cabinet (0-25) cylinder (0-7)	Because address compatible with BTSS
	Return via 14	

DSMIØRD	Set EDA bit = want in case Bit on.	Unique to Read
DSMJOIN	Store CAB, CYL, and pack # in EDA	common to read and write
DSMLETGO	Set Bits to Give up control. (in EDA)	Common to Read/Write and Routines wishing to end slice, then regain when I/O complete. I always re-enter of 'DASMMAIN'
	Leave DASM Force Int. and clock off. SVC 171.	
DSMIØRD	Set EDA = 'put back on disc'	unique to Write

DSMJOIN

DSMREWRT	Save important values. Set Drop/Add switch = 77 Lengths =	I have a matrix, 10 x 4 to save register 4 for each user.
	Are Lengths of data = . Yes	Am I rewriting an equal length record
	Nø	
	DSMRWADJ	
	Set Drop/Add switch = FF Lengths ≠	
DSMDELET	Flag index entry as deleted by X '09'	This is common to Rewrite where lengths are ≠, and deletes
	Write	Put index track back with deleted entry
GIVE UP CONTROL----------		
	DIAGNOSE	(OPTIONAL)

```
                Read index
                track back
                in.                          This is required to
                                             fool BTSS in order to
                                             another write.

                Restore all
                registers

                Was it
                a delete        Yes          If delete, I'm all
                                             done
N∅
                                DSMFEXIT     If a re-write,
                                             I go back and
                 DSMSTPNI                    track it like a write


DSMFEXIT        Set up
                standard
                error
                Bit

                Set Gr14
                to Main Line
                point that                   This is standard
                called function              return point

                  Return
                   via
                   14
```

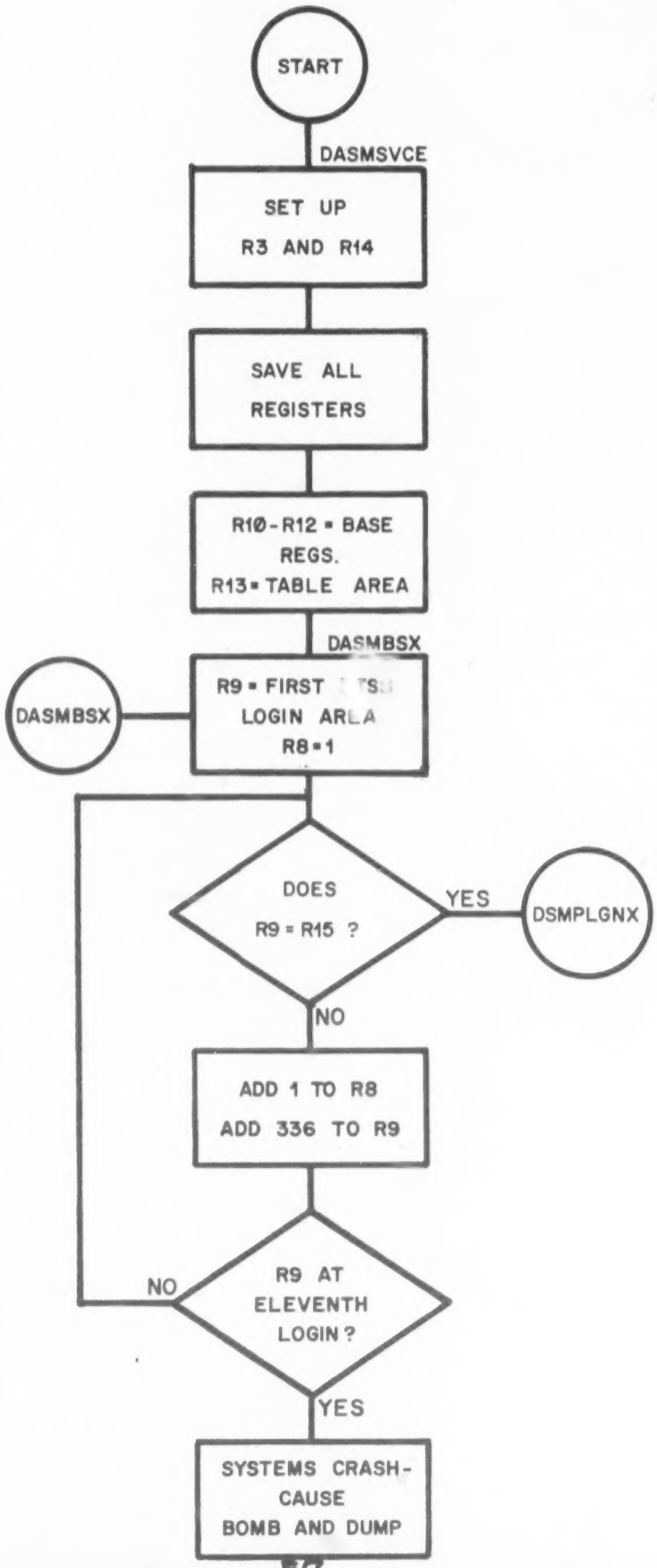
DSM Set 3	Set Gr_3 = start of HP given via Read/Write	Convert HP 3 in EDA to address
	Return via 14	
DSMRWADJ	Reduce piece count on 'ITH' By 1	This is done on Rewrites of = Length.

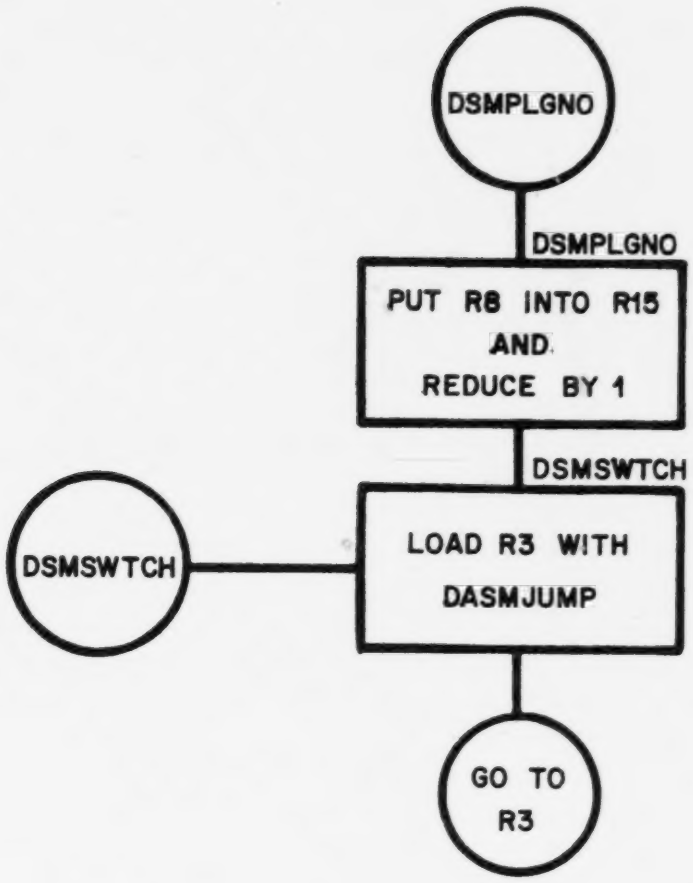
DSMWRTIT

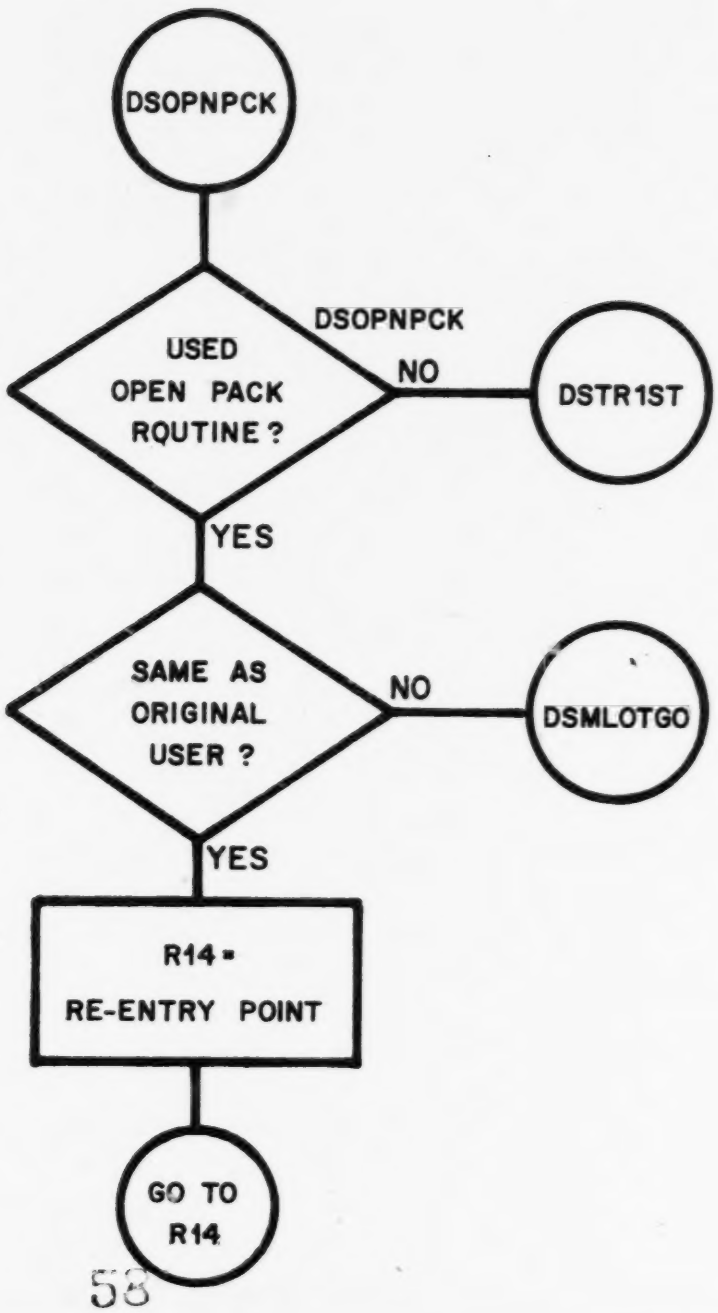
DASM APPENDIX J

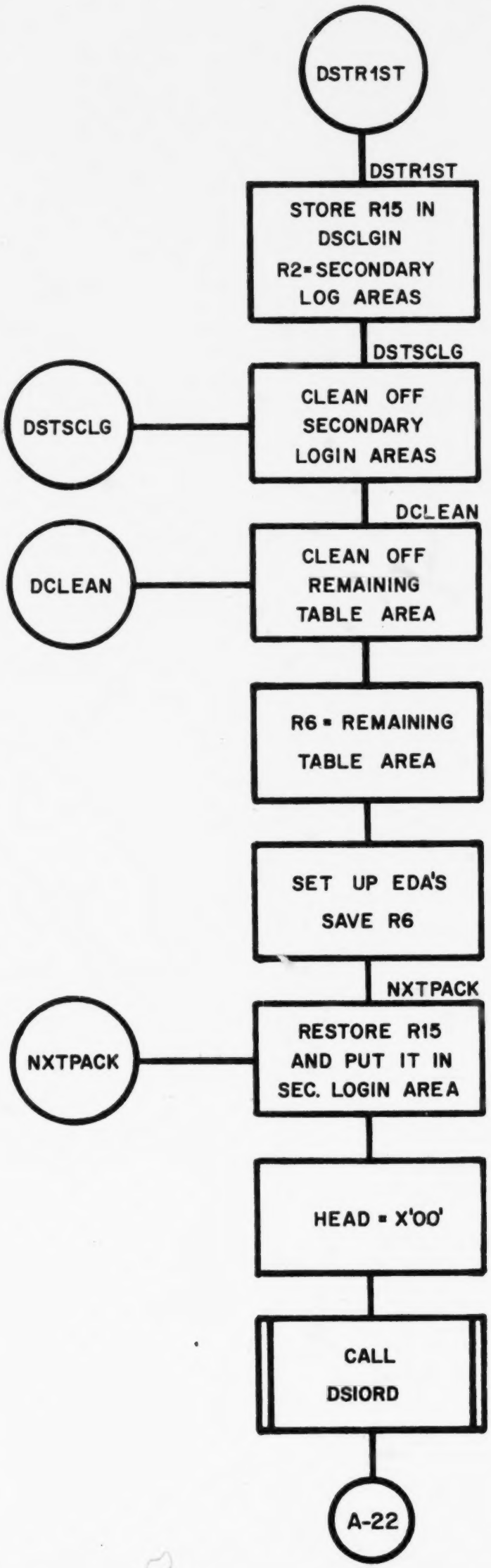
On-Line DASM (DASMSVCE)

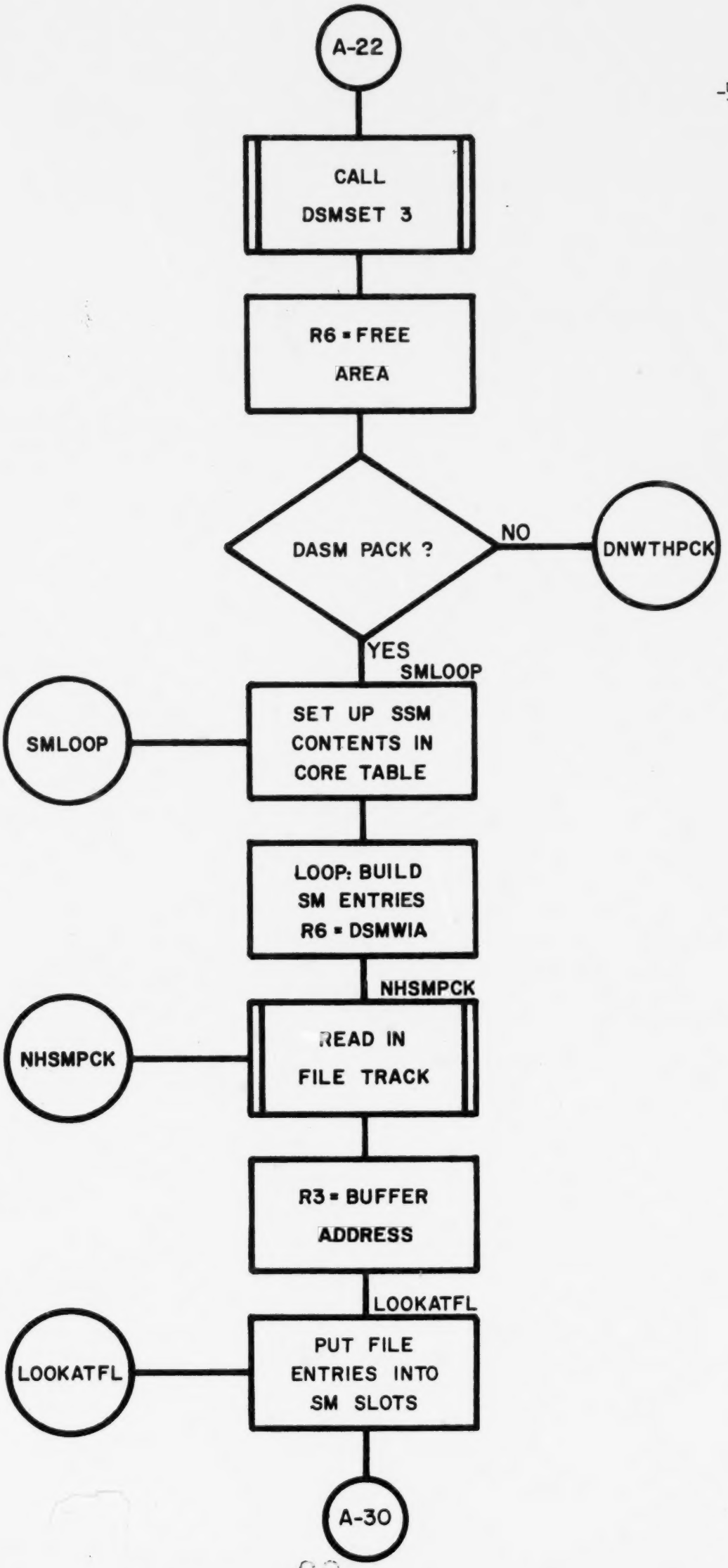
Flow Charts

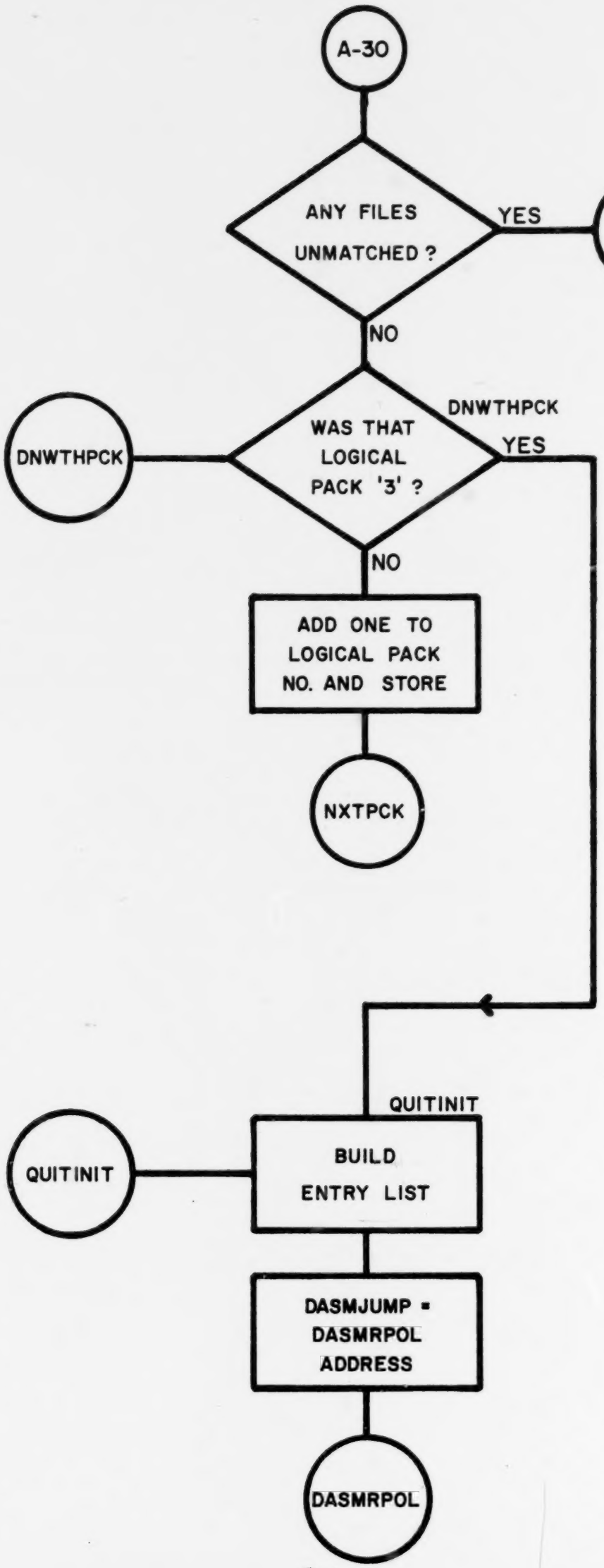

START
DASMSVCE
SET UP
R3 AND R14
SAVE ALL
REGISTERS
R10-R12 = BASE
REGS.
R13 = TABLE AREA
DASMBSX
DASMBSX
R9 = FIRST TSB
LOGIN AREA
R8 = 1
DOES
R9 = R15 ?
YES
DSMPLGNX
NO
ADD 1 TO R8
ADD 336 TO R9
NO
R9 AT
ELEVENTH
LOGIN ?
YES
SYSTEMS CRASH-
CAUSE
BOMB AND DUMP

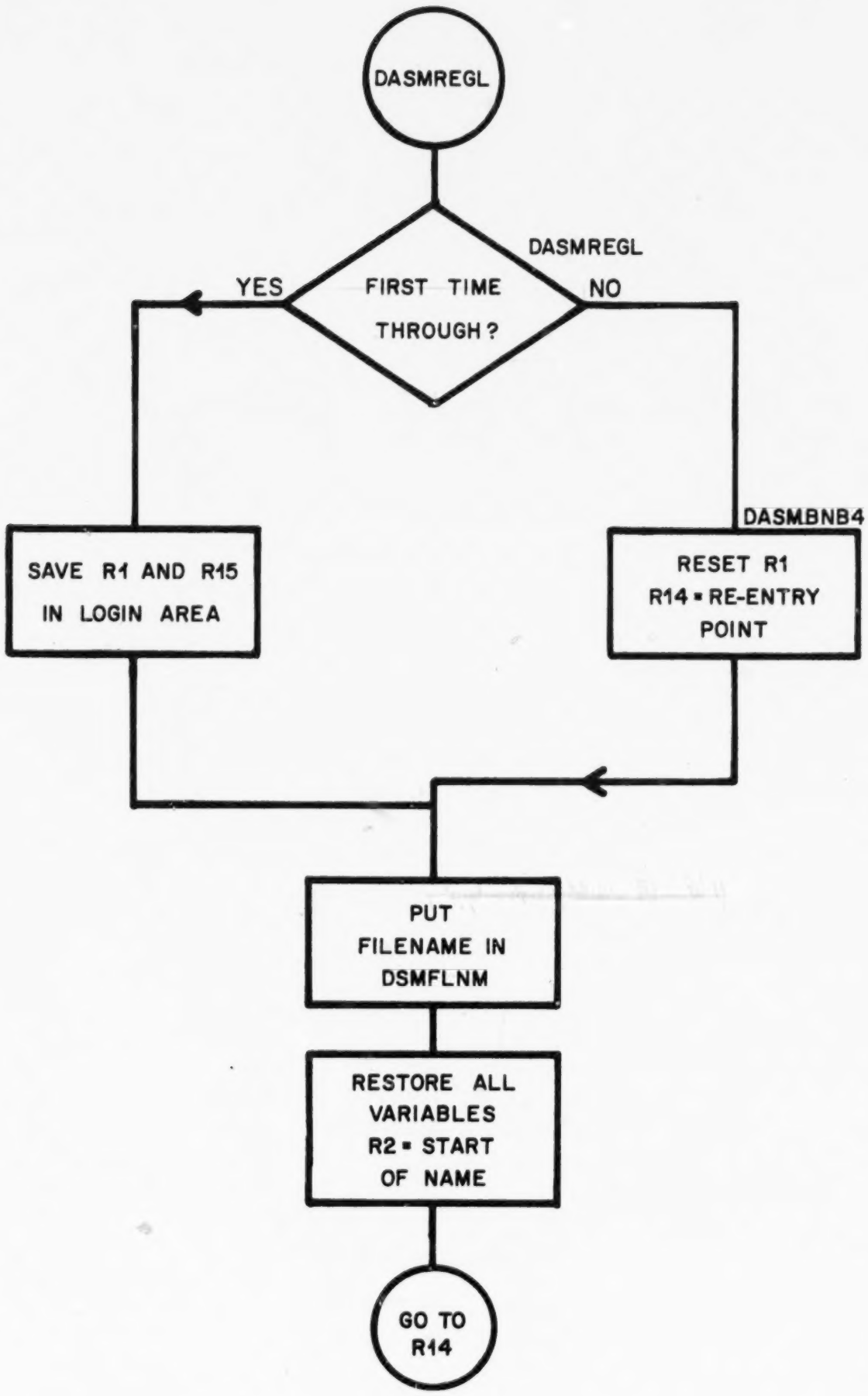

DSMPLGNO
DSMPLGNO
PUT R8 INTO R15
AND
REDUCE BY 1
DSMSWTCH
DSMSWTCH
LOAD R3 WITH
DASMJUMP
GO TO
R3

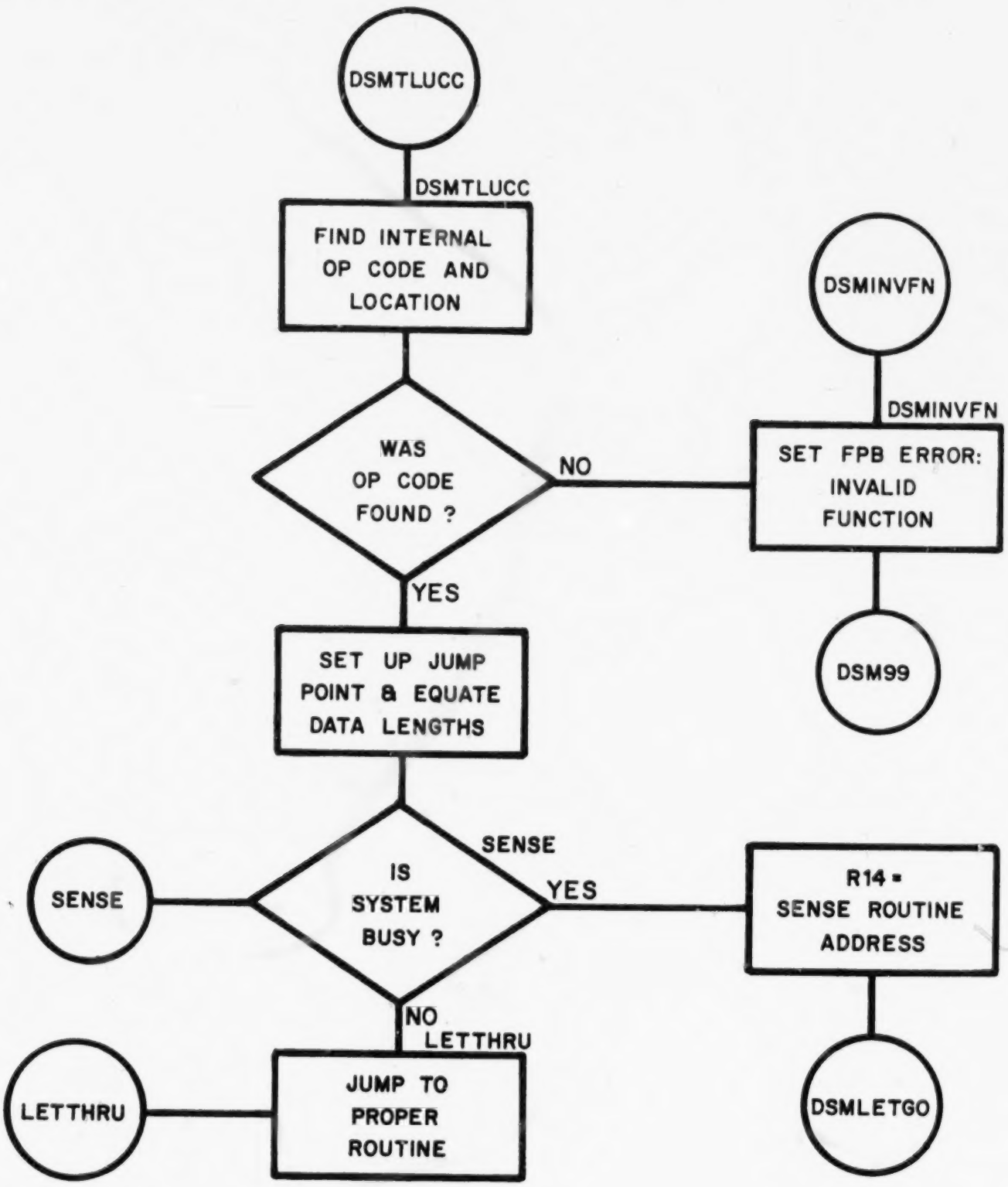

DSOPNPCK
DSOPNPCK
USED
OPEN PACK
RQUTINE ?
NO
DSTR1ST
YES
SAME AS
ORIGINAL
USER ?
NO
DSMLOTGO
YES
R14 =
RE-ENTRY POINT
GO TO
R14

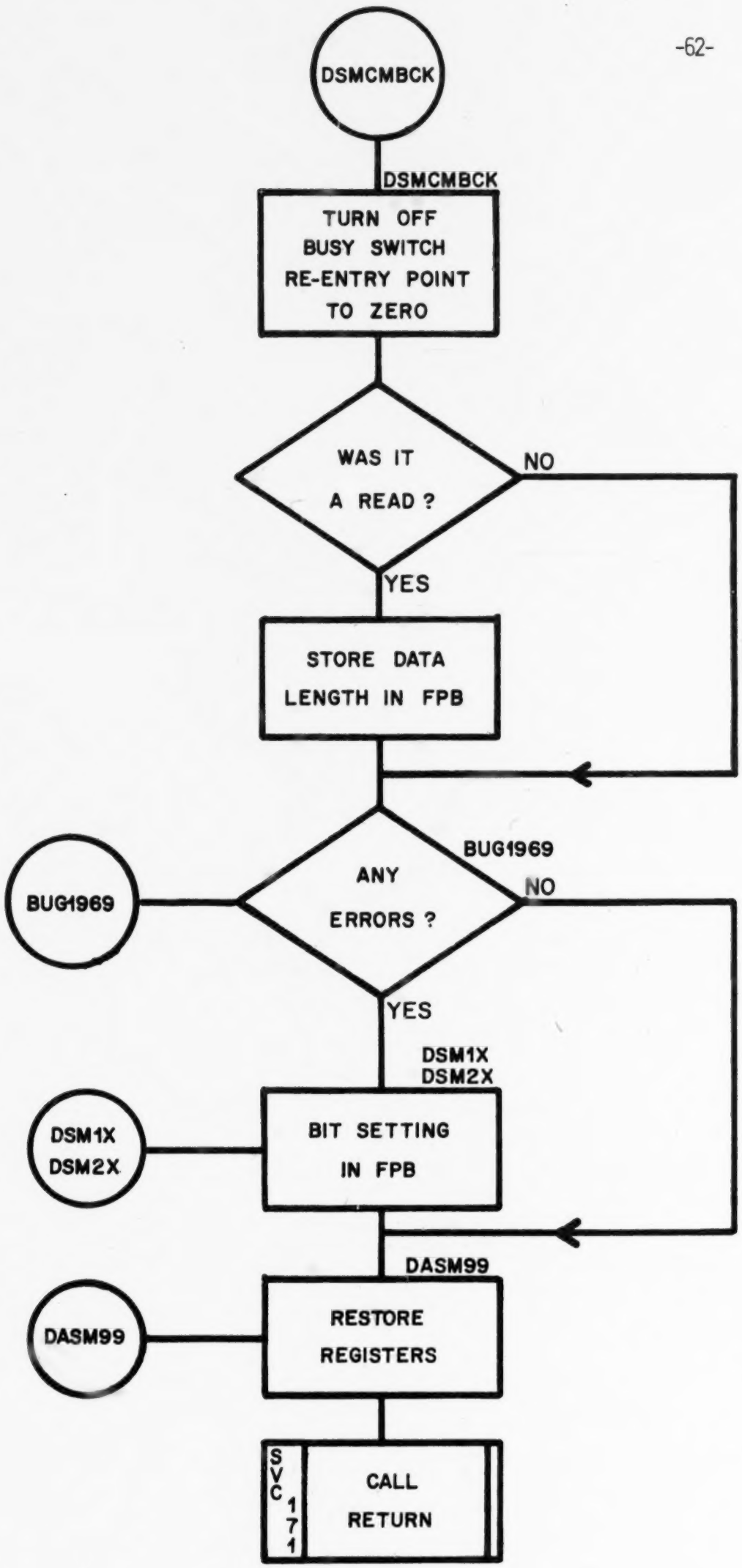

DSTR1ST
DSTR1ST
STORE R15 IN DSCLGIN R2=SECONDARY LOG AREAS
DSTSCLG
DSTSCLG
CLEAN OFF SECONDARY LOGIN AREAS
DCLEAN
DCLEAN
CLEAN OFF REMAINING TABLE AREA
R6 = REMAINING TABLE AREA
SET UP EDA'S SAVE R6
NXTPACK
NXTPACK
RESTORE R15 AND PUT IT IN SEC. LOGIN AREA
HEAD = X'00'
CALL DSIORD
A-22

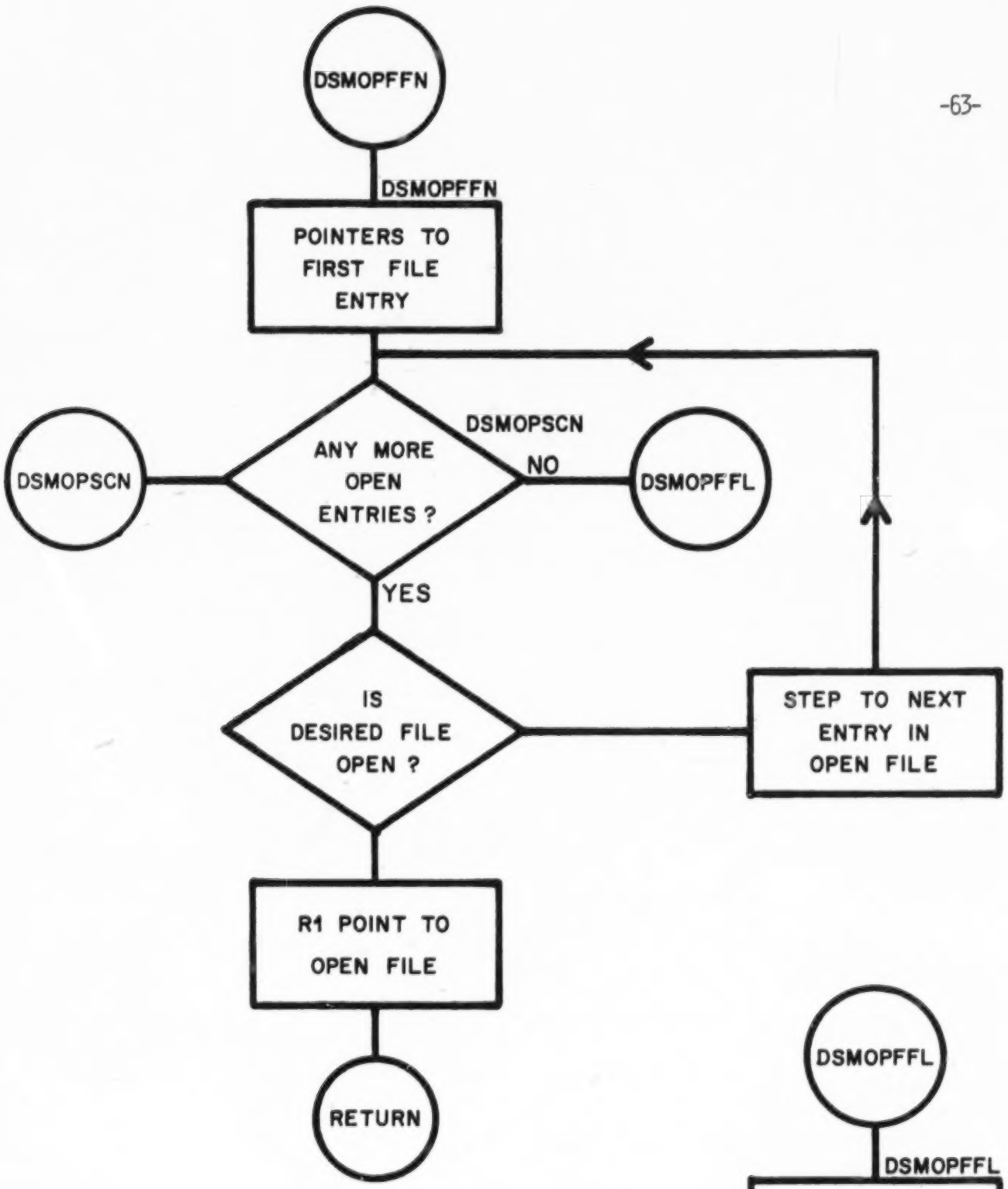

A-22
CALL
DSMSET 3
R6 = FREE
AREA
DASM PACK ?
NO
DNWTHPCK
YES
SMLOOP
SMLOOP
SET UP SSM
CONTENTS IN
CORE TABLE
LOOP: BUILD
SM ENTRIES
R6 = DSMWIA
NHSMPCK
NHSMPCK
READ IN
FILE TRACK
R3 = BUFFER
ADDRESS
LOOKATFL
LOOKATFL
PUT FILE
ENTRIES INTO
SM SLOTS
A-30

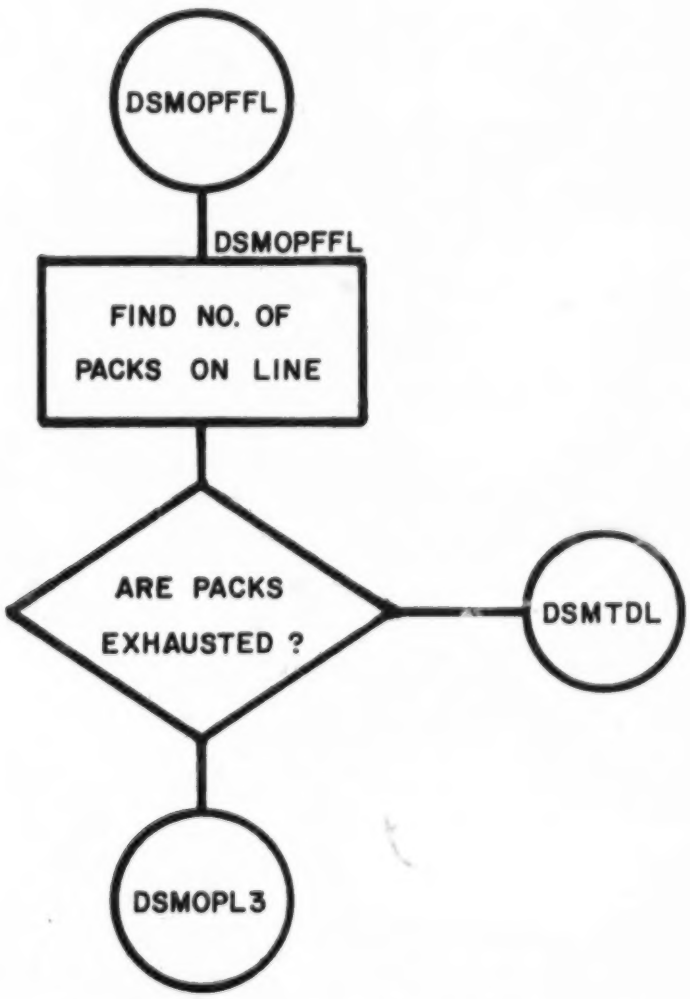

A-30
ANY FILES UNMATCHED ?
YES
BOMB
NO
WAS THAT LOGICAL PACK '3' ?
DNWTHPCK
YES
DNWTHPCK
NO
ADD ONE TO LOGICAL PACK NO. AND STORE
NXTPCK
QUITINIT
BUILD ENTRY LIST
QUITINIT
DASMJUMP = DASMRPOL ADDRESS
DASMRPOL

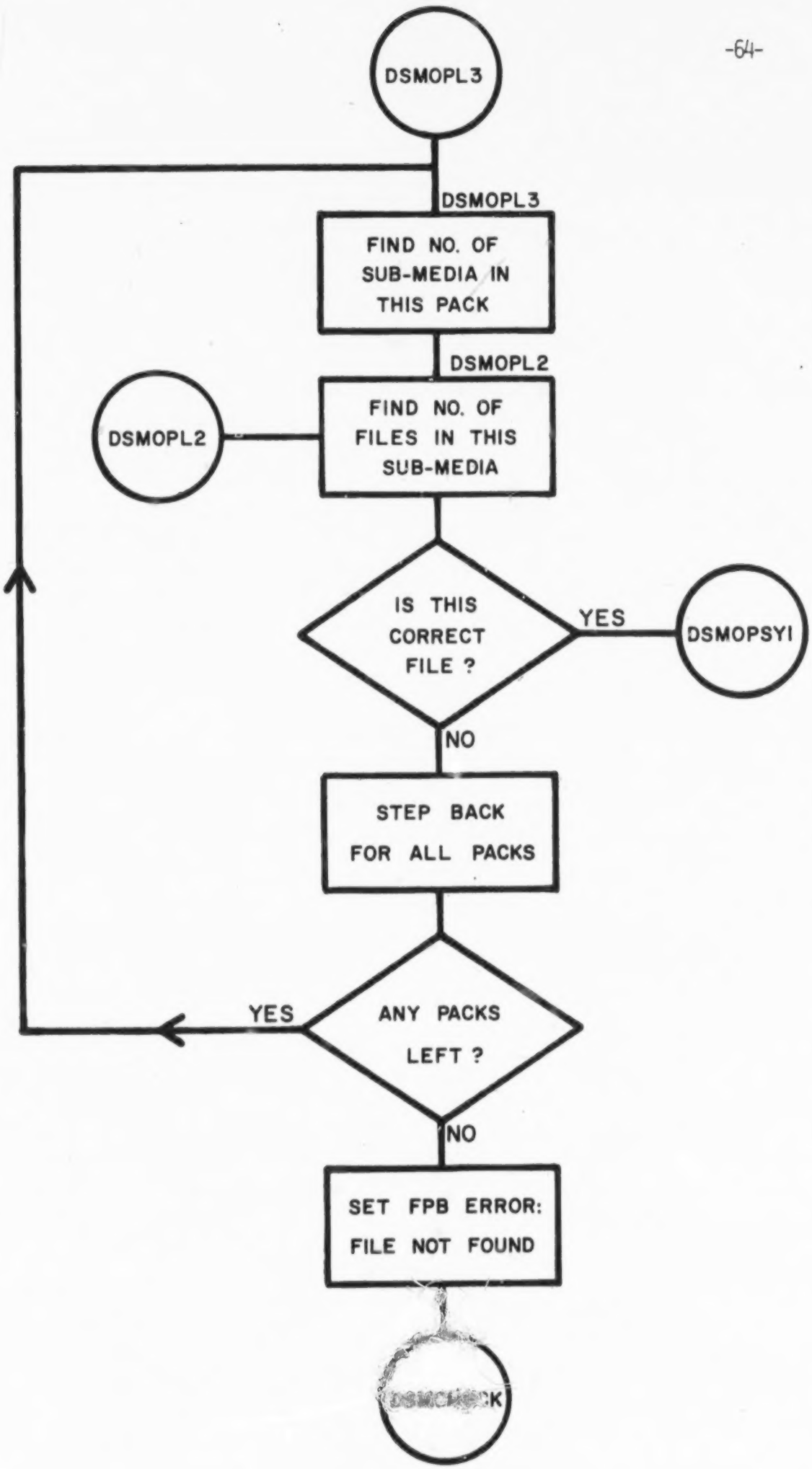

DASMREGL
DASMREGL
YES
FIRST TIME THROUGH?
NO
DASMBNB4
SAVE R1 AND R15 IN LOGIN AREA
RESET R1
R14 = RE-ENTRY POINT
PUT FILENAME IN DSMFLNM
RESTORE ALL VARIABLES
R2 = START OF NAME
GO TO R14

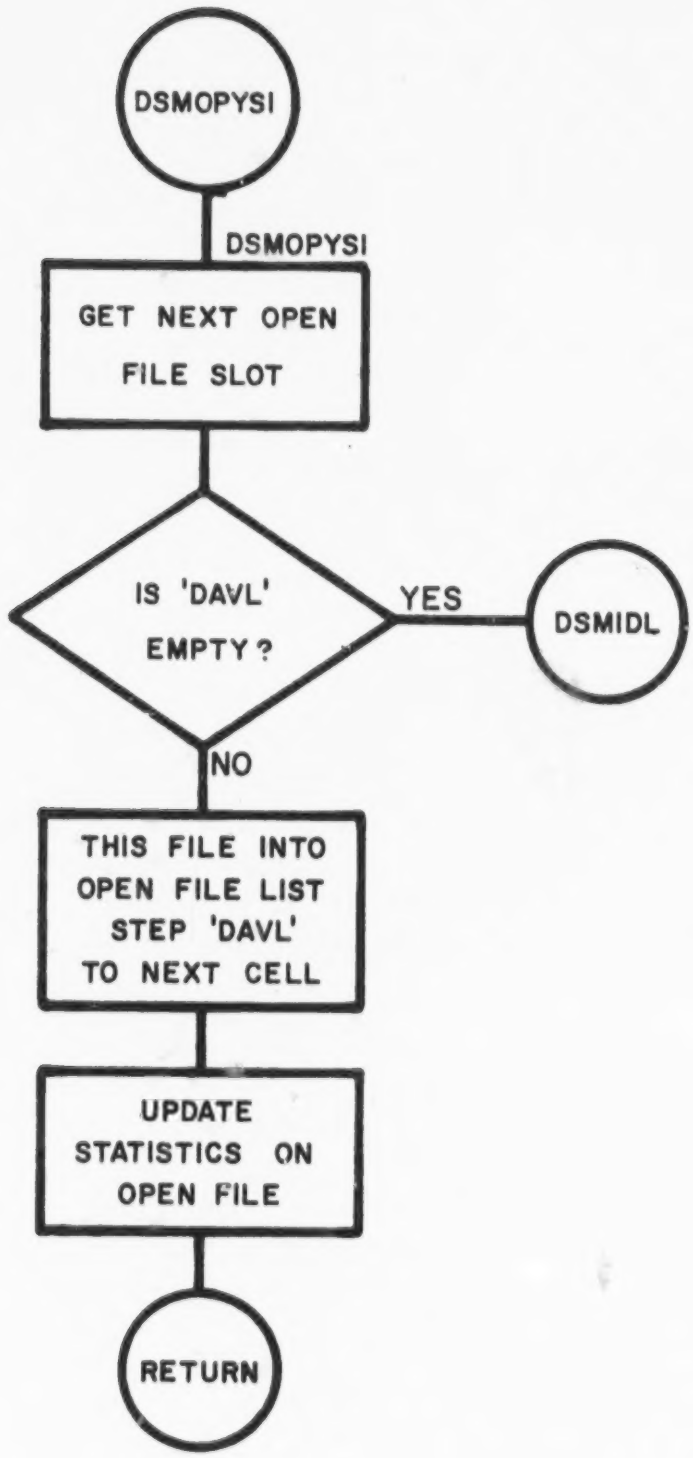

DSMTLUCC
DSMTLUCC
FIND INTERNAL OP CODE AND LOCATION
WAS OP CODE FOUND ?
NO
YES
DSMINVFN
DSMINVFN
SET FPB ERROR: INVALID FUNCTION
DSM99
SET UP JUMP POINT & EQUATE DATA LENGTHS
SENSE
IS SYSTEM BUSY ?
SENSE
YES
SENSE
R14 = SENSE ROUTINE ADDRESS
NO
LETTHRU
LETTHRU
JUMP TO PROPER ROUTINE
DSMLETGO

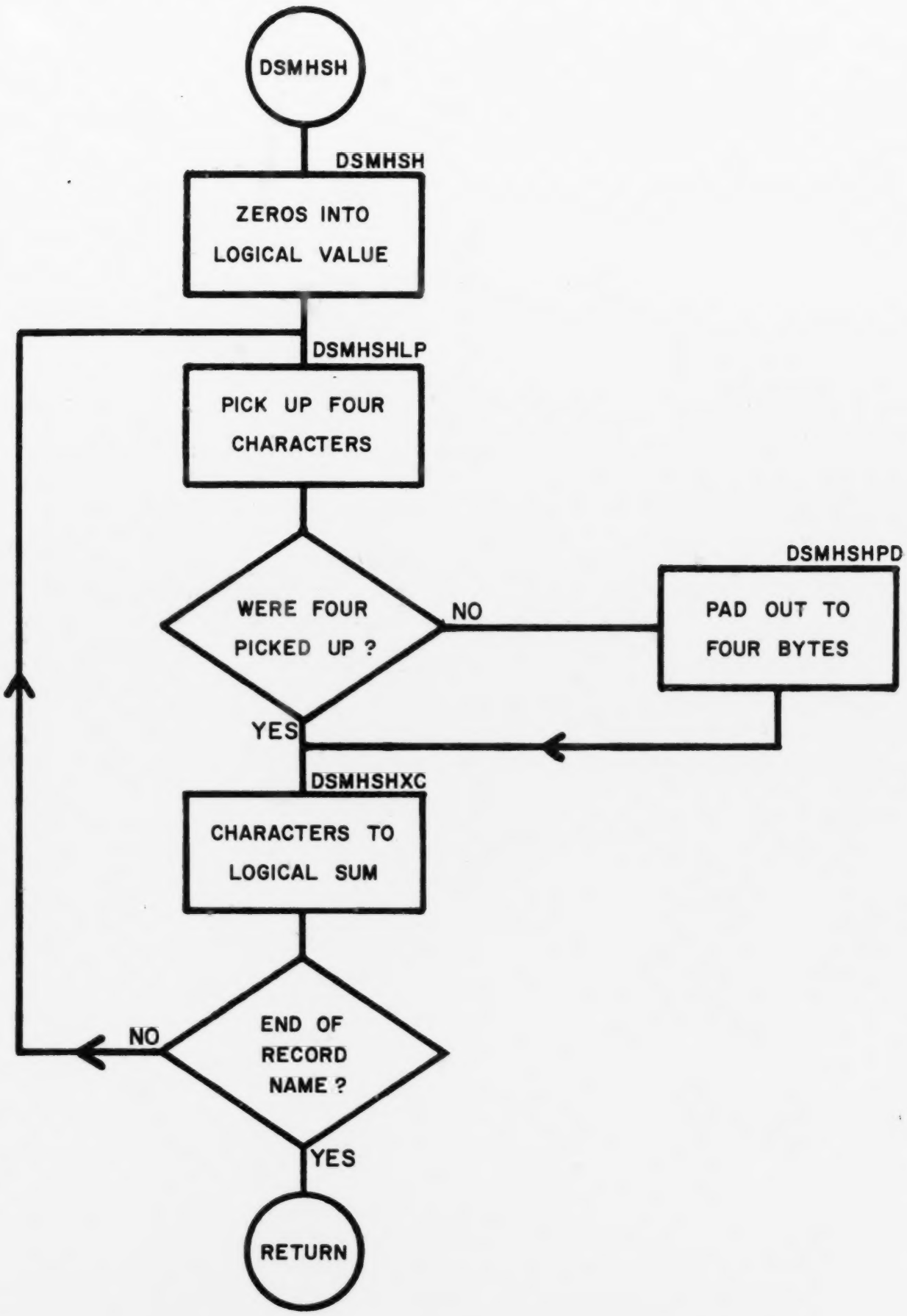

DSMCMBCK
DSMCMBCK
TURN OFF
BUSY SWITCH
RE-ENTRY POINT
TO ZERO
WAS IT
A READ ?
NO
YES
STORE DATA
LENGTH IN FPB
BUG1969
ANY
ERRORS ?
BUG1969
NO
YES
DSM1X
DSM2X
DSM1X
DSM2X
BIT SETTING
IN FPB
DASM99
DASM99
RESTORE
REGISTERS
SVC 171
CALL
RETURN

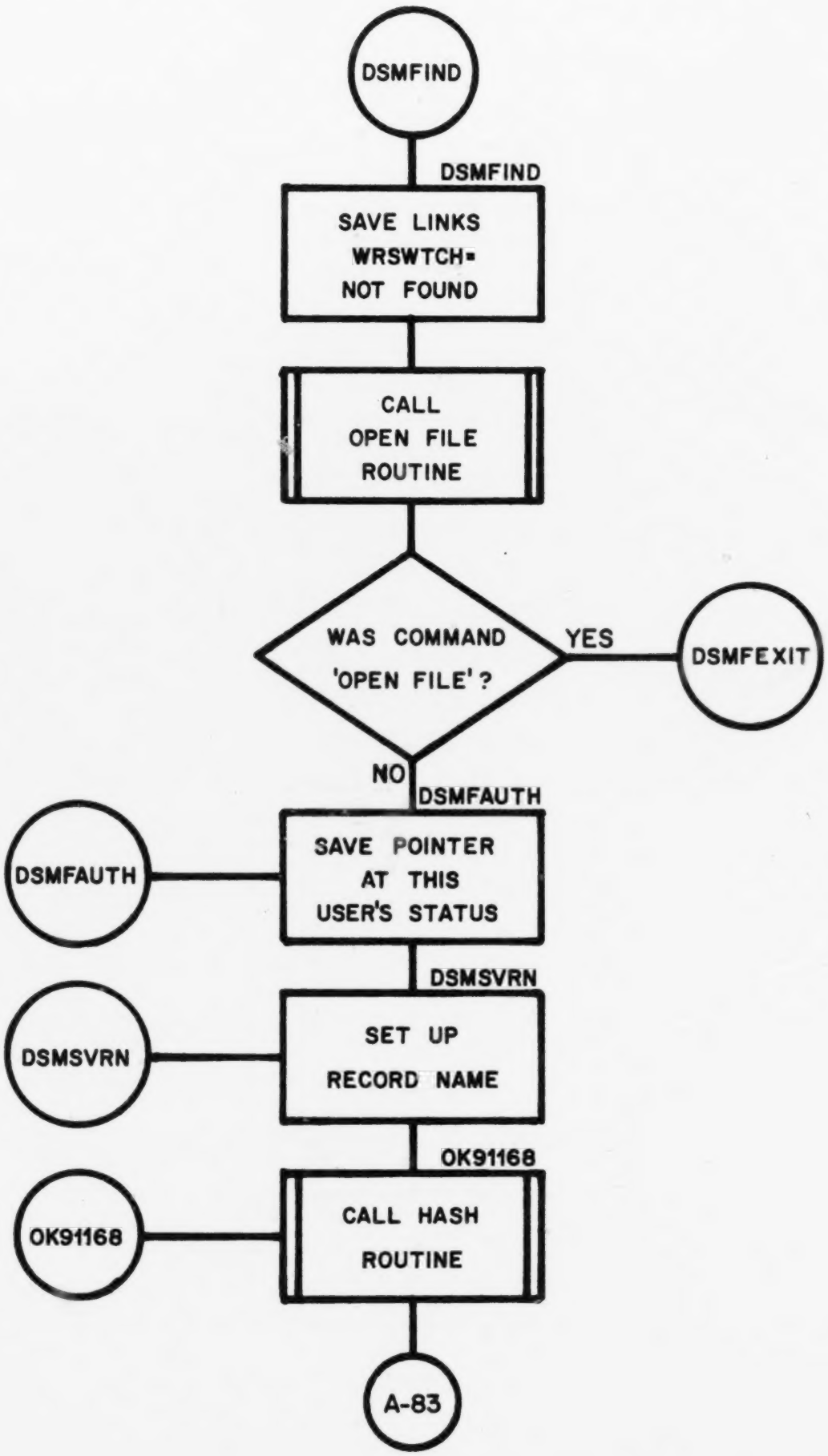

DSMOPFFN
DSMOPFFN
POINTERS TO FIRST FILE ENTRY
ANY MORE OPEN ENTRIES ?
DSMOPSCN
DSMOPSCN
NO
DSMOPFFL
YES
IS DESIRED FILE OPEN ?
STEP TO NEXT ENTRY IN OPEN FILE
R1 POINT TO OPEN FILE
RETURN

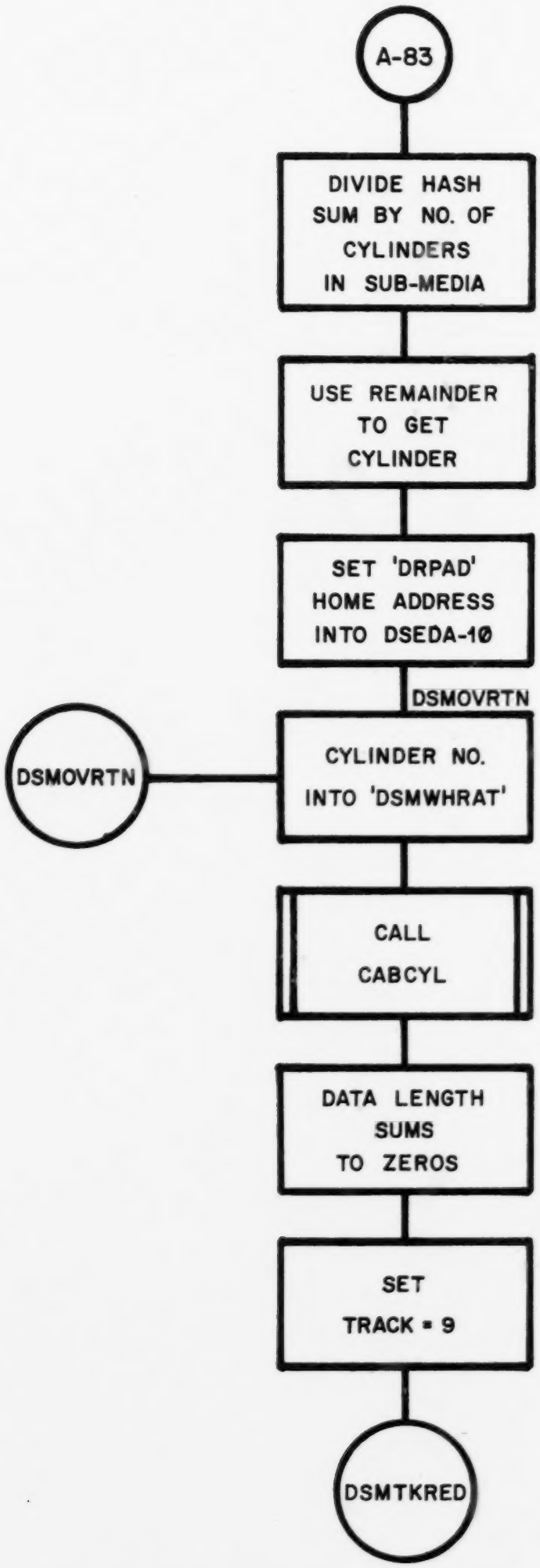

DSMOPFFL
DSMOPFFL
FIND NO. OF PACKS ON LINE
ARE PACKS EXHAUSTED ?
DSMTDL
DSMOPL3

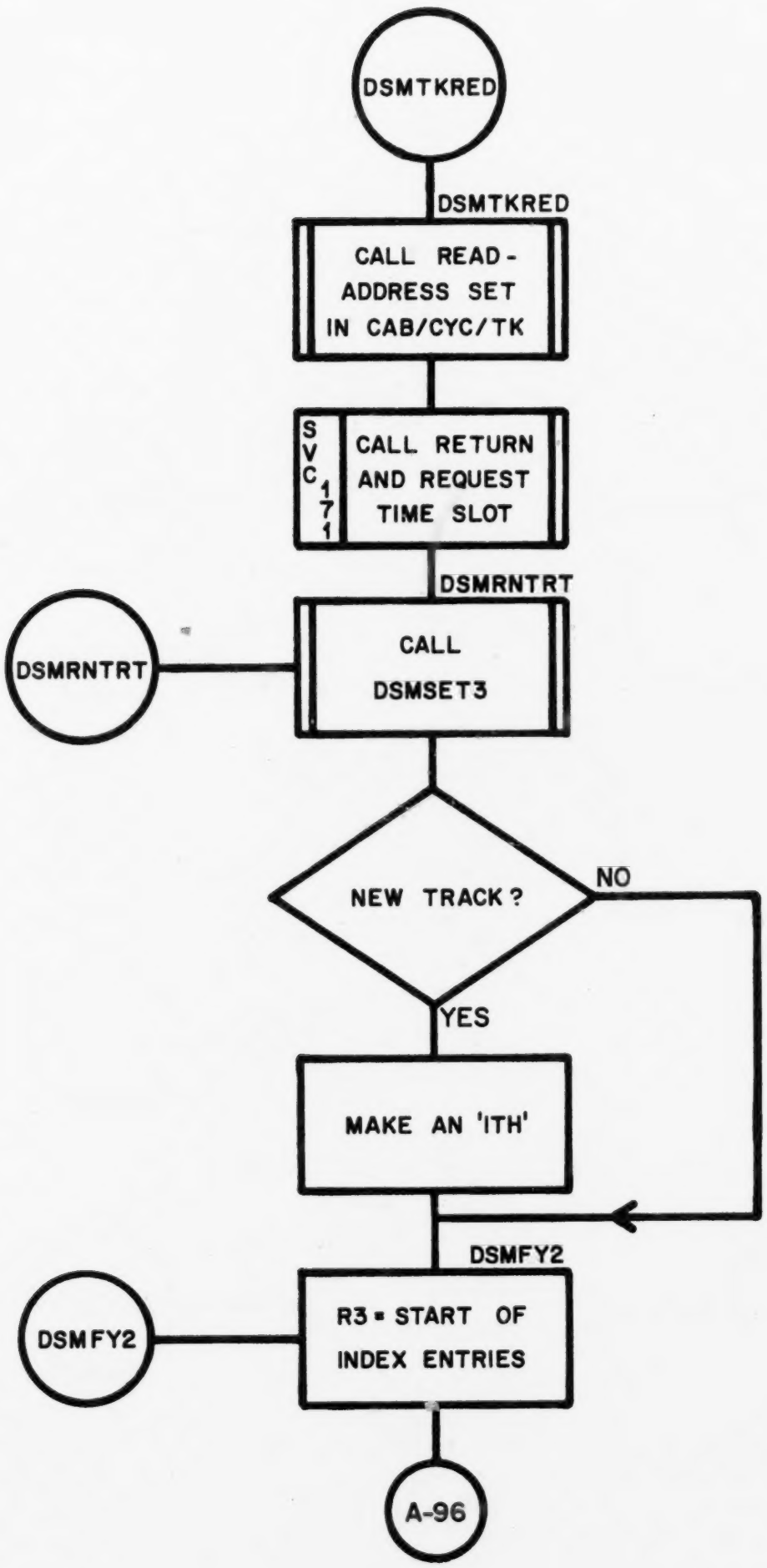

DSMOPL3
DSMOPL3
FIND NO. OF SUB-MEDIA IN THIS PACK
DSMOPL2
DSMOPL2
FIND NO. OF FILES IN THIS SUB-MEDIA
IS THIS CORRECT FILE ?
YES
DSMOPSYI
NO
STEP BACK FOR ALL PACKS
ANY PACKS LEFT ?
YES
NO
SET FPB ERROR: FILE NOT FOUND

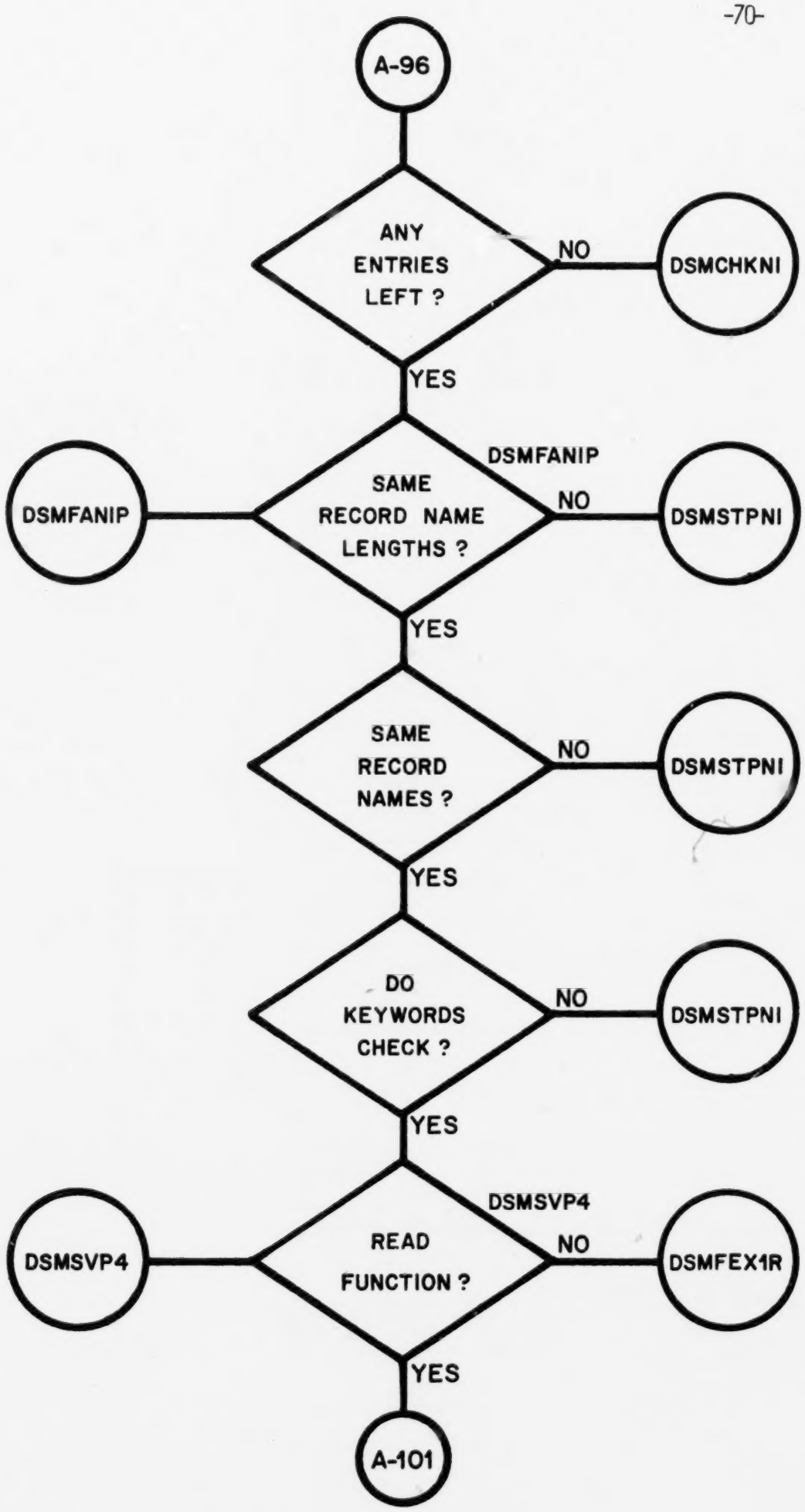

DSMOPYSI
DSMOPYSI
GET NEXT OPEN FILE SLOT
IS 'DAVL' EMPTY?
YES
DSMIDL
NO
THIS FILE INTO OPEN FILE LIST STEP 'DAVL' TO NEXT CELL
UPDATE STATISTICS ON OPEN FILE
RETURN

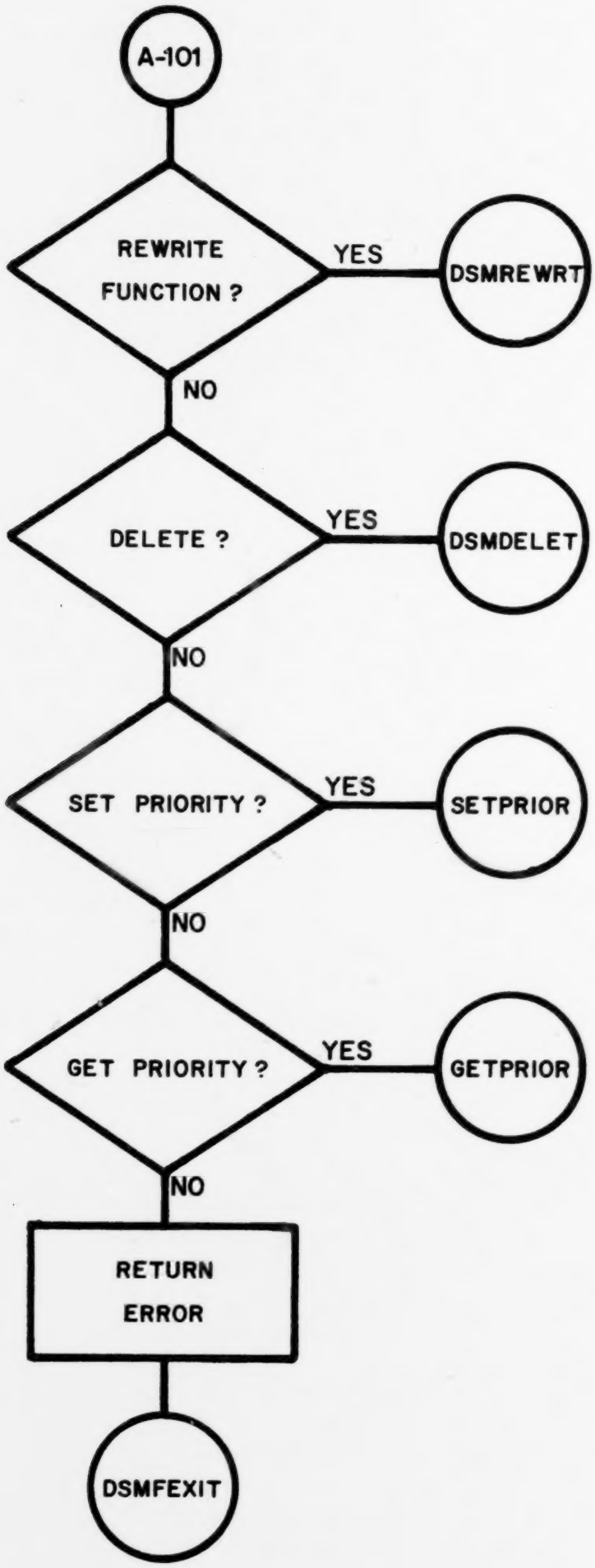

DSMHSH
DSMHSH
ZEROS INTO
LOGICAL VALUE
DSMHSHLP
PICK UP FOUR
CHARACTERS
WERE FOUR
PICKED UP ?
NO
DSMHSHPD
PAD OUT TO
FOUR BYTES
YES
DSMHSHXC
CHARACTERS TO
LOGICAL SUM
END OF
RECORD
NAME ?
NO
YES
RETURN

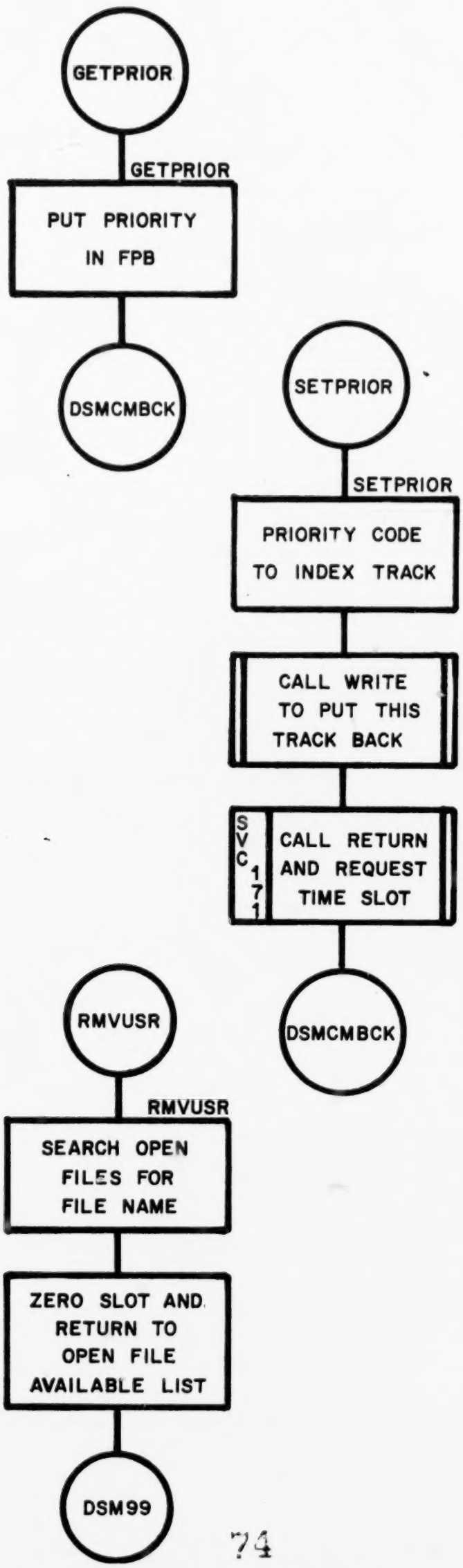

DSMFIND
DSMFIND
SAVE LINKS
WRSWTCH=
NOT FOUND
CALL
OPEN FILE
ROUTINE
WAS COMMAND
'OPEN FILE'?
YES
DSMFEXIT
NO
DSMFAUTH
DSMFAUTH
SAVE POINTER
AT THIS
USER'S STATUS
DSMSVRN
DSMSVRN
SET UP
RECORD NAME
OK91168
OK91168
CALL HASH
ROUTINE
A-83

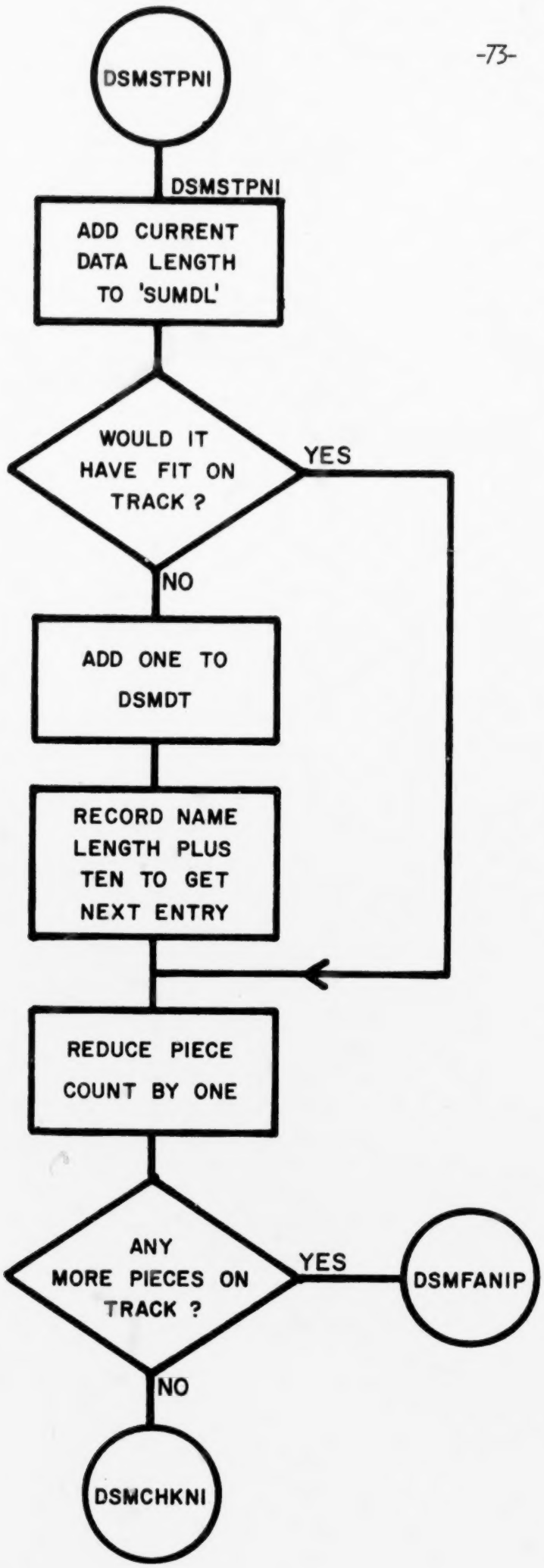

A-83
DIVIDE HASH SUM BY NO. OF CYLINDERS IN SUB-MEDIA
USE REMAINDER TO GET CYLINDER
SET 'DRPAD' HOME ADDRESS INTO DSEDA-10
DSMOVRTN
DSMOVRTN
CYLINDER NO. INTO 'DSMWHRAT'
CALL CABCYL
DATA LENGTH SUMS TO ZEROS
SET TRACK = 9
DSMTKRED

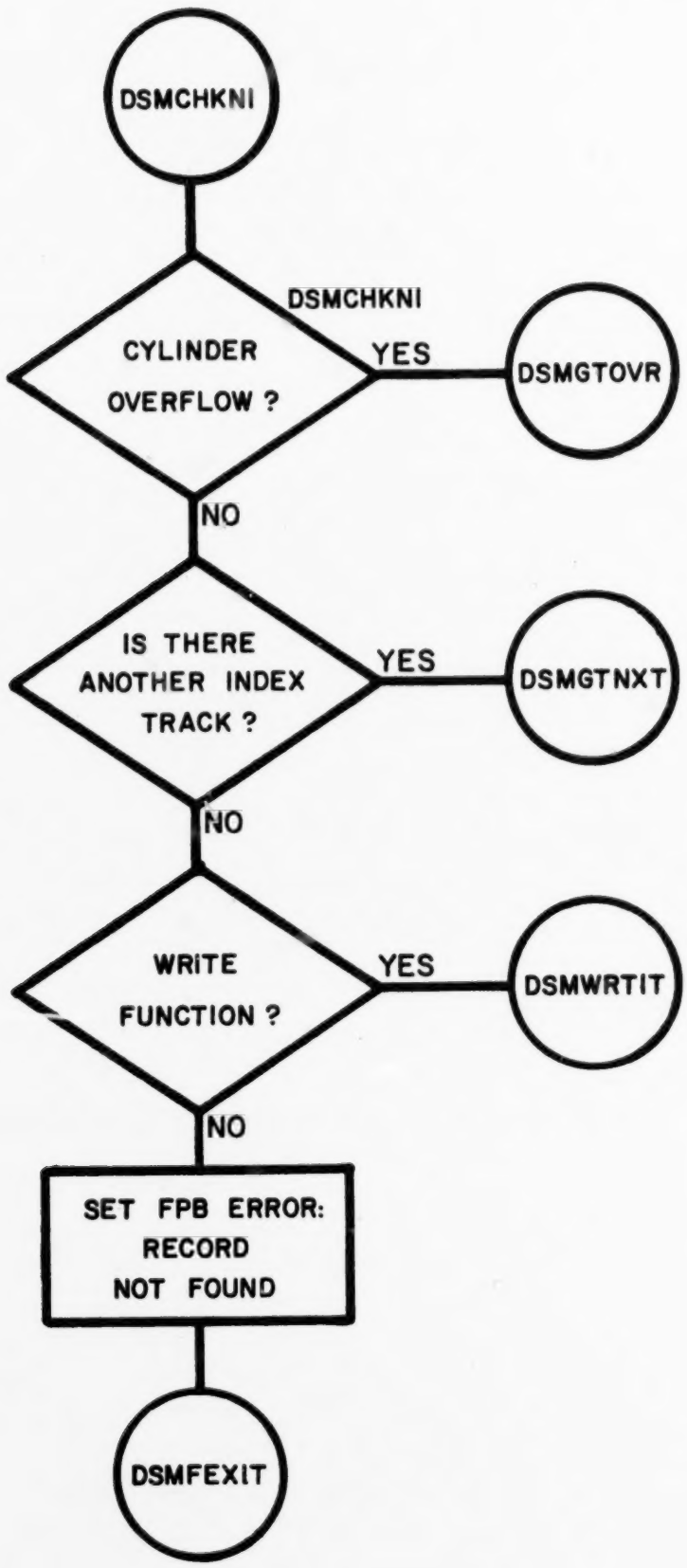

DSMTKRED
DSMTKRED
CALL READ-ADDRESS SET IN CAB/CYC/TK
SVC 171
CALL RETURN AND REQUEST TIME SLOT
DSMRNTRT
DSMRNTRT
CALL DSMSET3
NEW TRACK?
NO
YES
MAKE AN 'ITH'
DSMFY2
DSMFY2
R3 = START OF INDEX ENTRIES
A-96

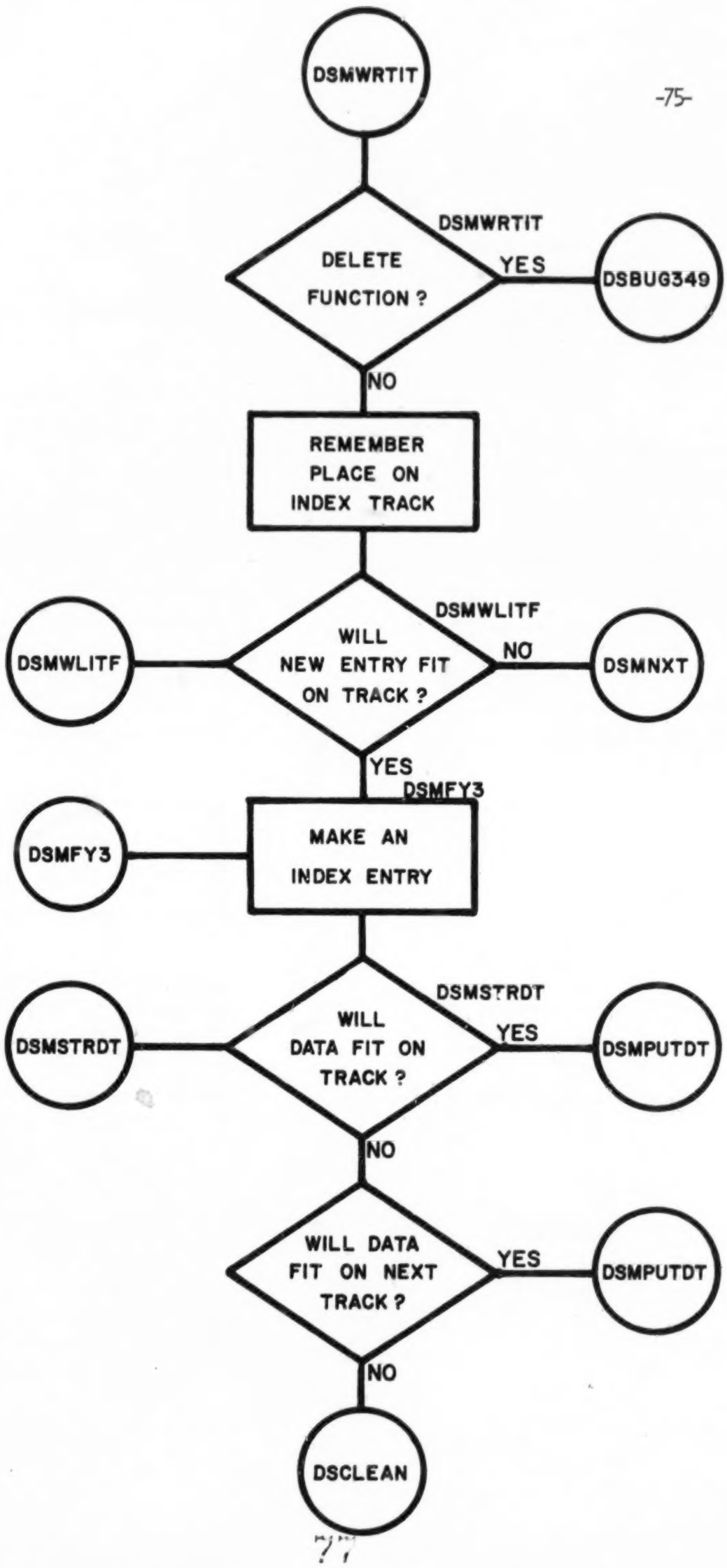

A-96
ANY ENTRIES LEFT ?
NO
DSMCHKNI
YES
DSMFANIP
SAME RECORD NAME LENGTHS ?
DSMFANIP
NO
DSMSTPNI
YES
SAME RECORD NAMES ?
NO
DSMSTPNI
YES
DO KEYWORDS CHECK ?
NO
DSMSTPNI
YES
DSMSVP4
READ FUNCTION ?
DSMSVP4
NO
DSMFEX1R
YES
A-101

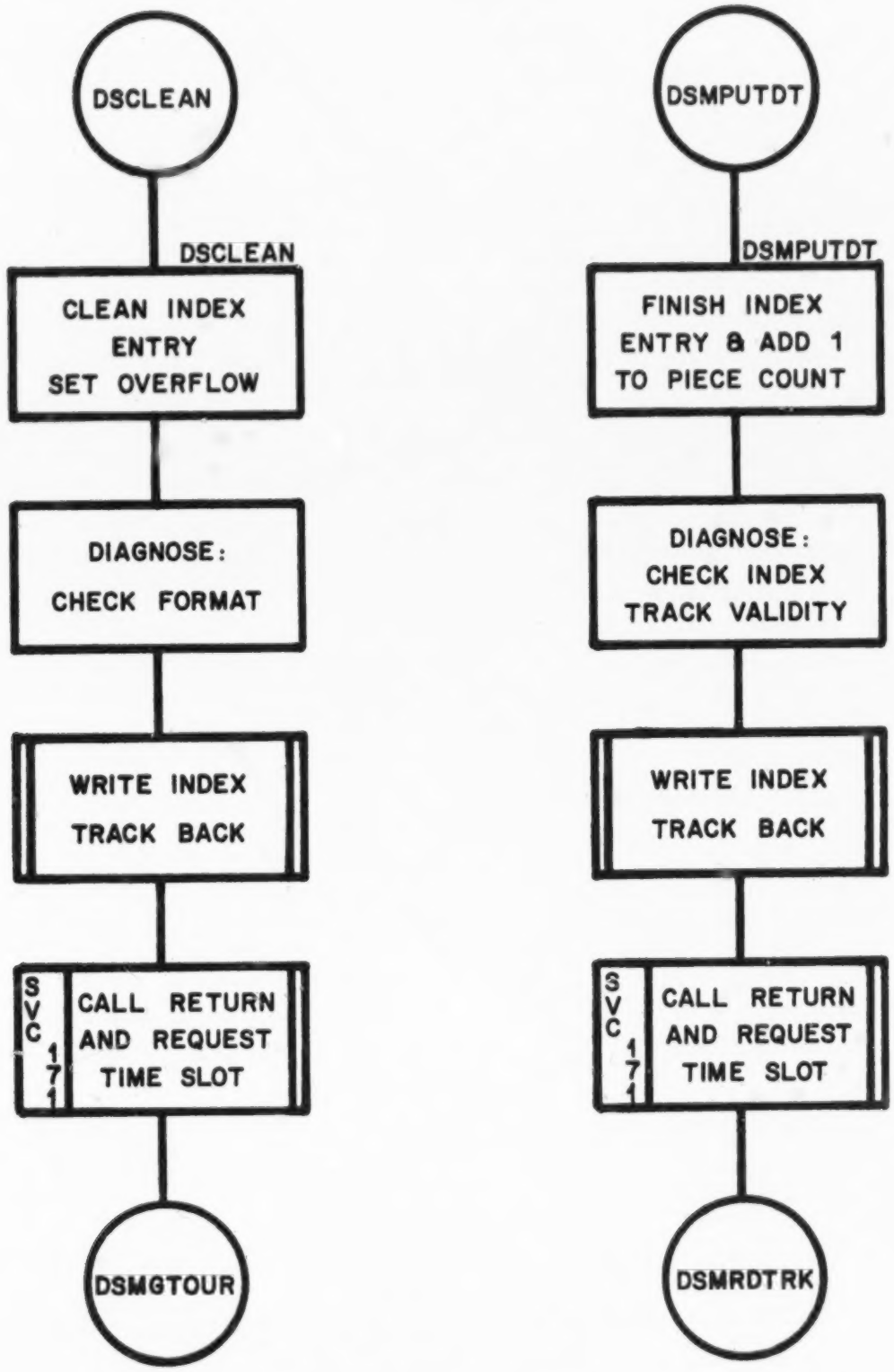

A-101
REWRITE FUNCTION ?
YES
DSMREWRT
NO
DELETE ?
YES
DSMDELET
NO
SET PRIORITY ?
YES
SETPRIOR
NO
GET PRIORITY ?
YES
GETPRIOR
NO
RETURN ERROR
DSMFEXIT

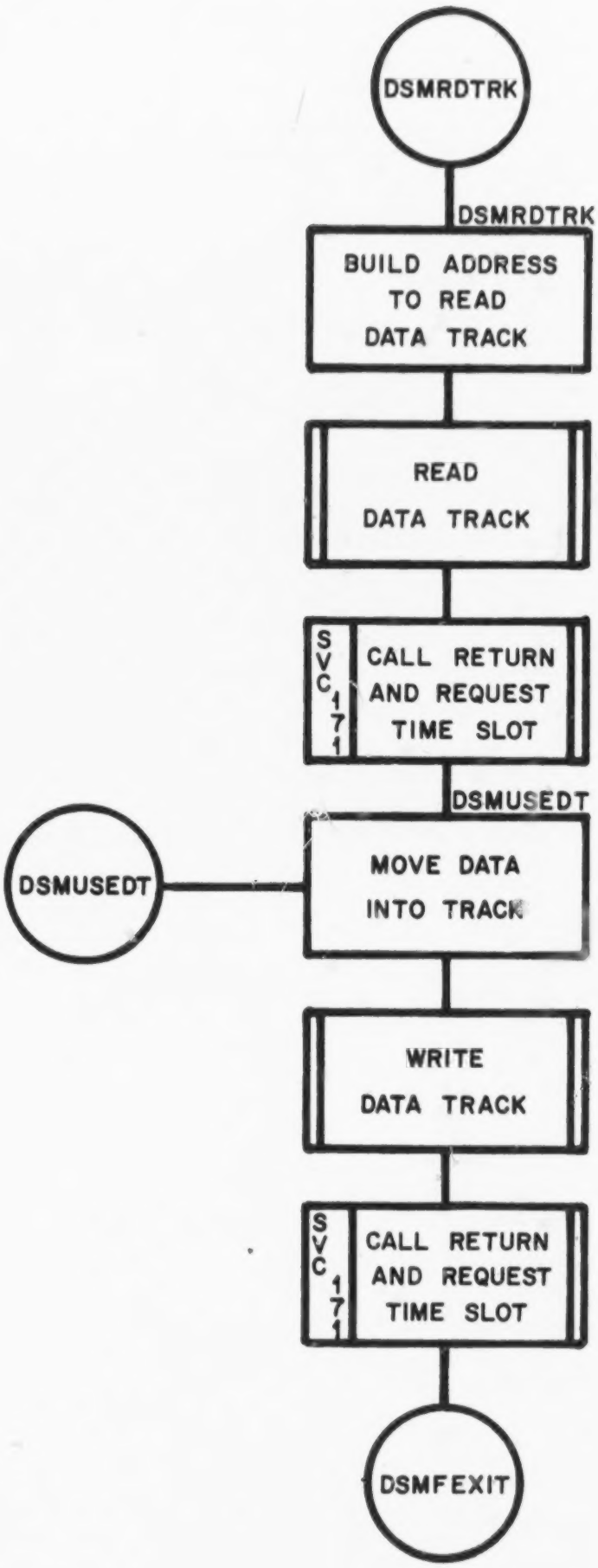

GETPRIOR
GETPRIOR
PUT PRIORITY IN FPB
DSMCMBCK
SETPRIOR
SETPRIOR
PRIORITY CODE TO INDEX TRACK
CALL WRITE TO PUT THIS TRACK BACK
SVC 171
CALL RETURN AND REQUEST TIME SLOT
DSMCMBCK
RMVUSR
RMVUSR
SEARCH OPEN FILES FOR FILE NAME
ZERO SLOT AND RETURN TO OPEN FILE AVAILABLE LIST
DSM99

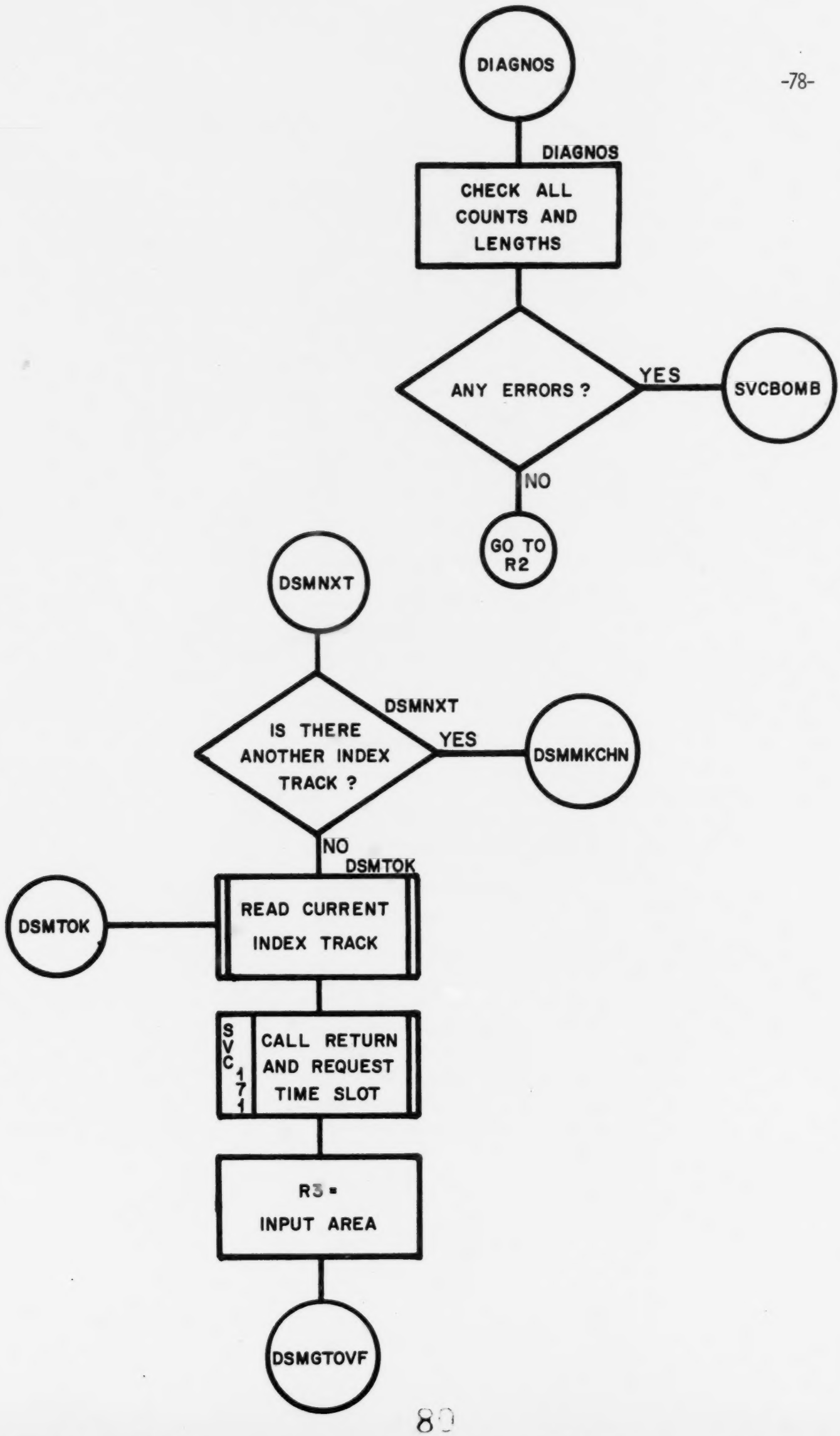

DSMSTPNI
DSMSTPNI
ADD CURRENT DATA LENGTH TO 'SUMDL'
WOULD IT HAVE FIT ON TRACK ?
YES
NO
ADD ONE TO DSMDT
RECORD NAME LENGTH PLUS TEN TO GET NEXT ENTRY
REDUCE PIECE COUNT BY ONE
ANY MORE PIECES ON TRACK ?
YES
DSMFANIP
NO
DSMCHKNI

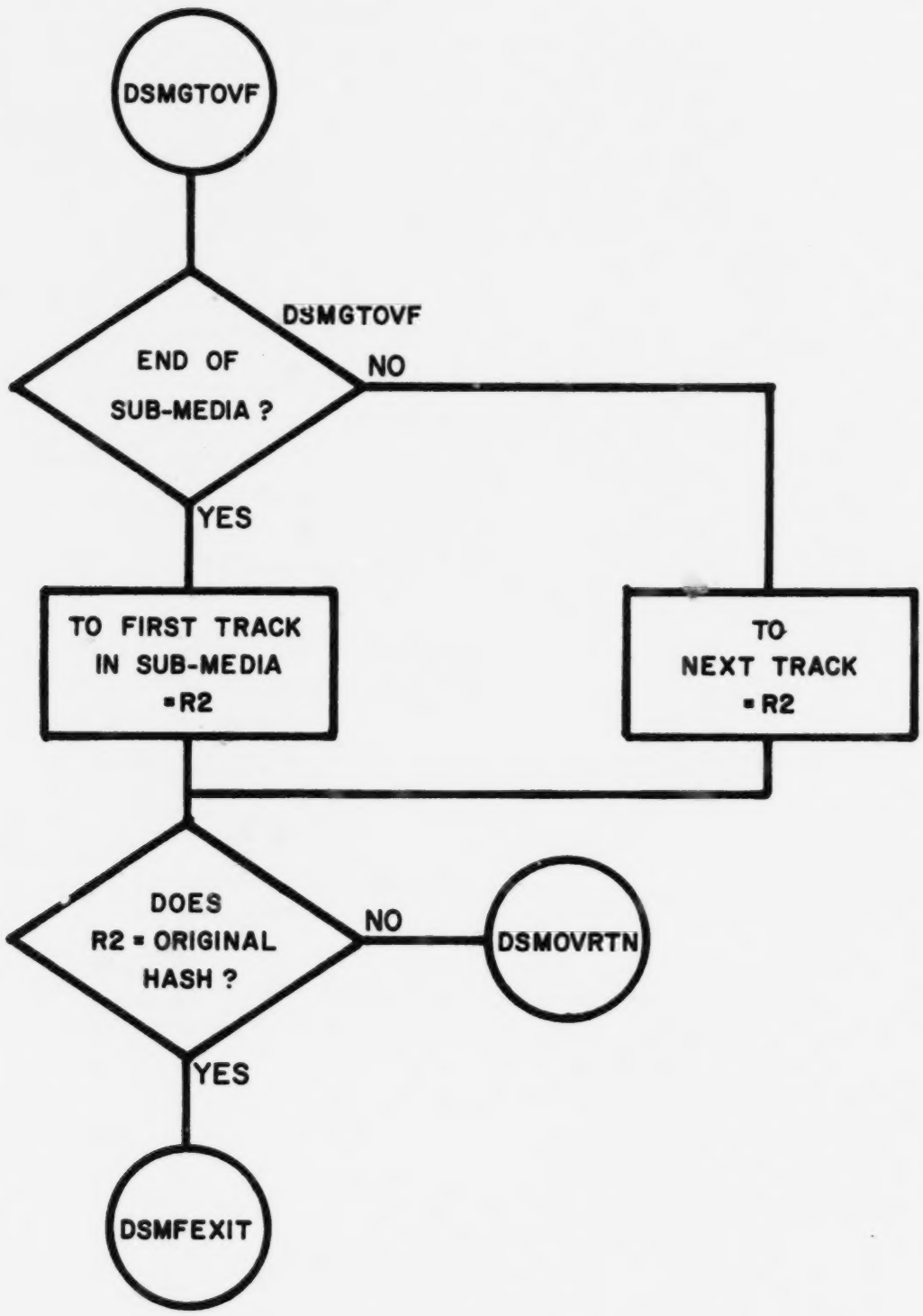

DSMCHKNI
DSMCHKNI
CYLINDER OVERFLOW ?
YES
DSMGTOVR
NO
IS THERE ANOTHER INDEX TRACK ?
YES
DSMGTNXT
NO
WRITE FUNCTION ?
YES
DSMWRTIT
NO
SET FPB ERROR: RECORD NOT FOUND
DSMFEXIT


DSMWRTIT
DSMWRTIT
DELETE FUNCTION ?
YES
DSBUG349
NO
REMEMBER PLACE ON INDEX TRACK
DSMWLITF
WILL NEW ENTRY FIT ON TRACK ?
NO
DSMNXT
DSMWLITF
YES
DSMFY3
DSMFY3
MAKE AN INDEX ENTRY
DSMSTRDT
WILL DATA FIT ON TRACK ?
YES
DSMPUTDT
DSMSTRDT
NO
WILL DATA FIT ON NEXT TRACK ?
YES
DSMPUTDT
NO
DSCLEAN

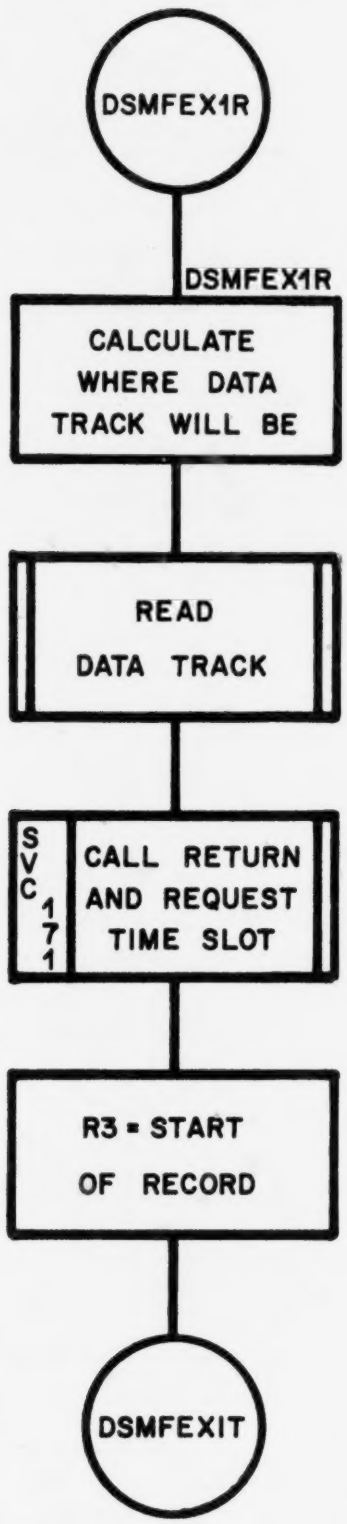

DSCLEAN
DSCLEAN
CLEAN INDEX ENTRY SET OVERFLOW
DIAGNOSE: CHECK FORMAT
WRITE INDEX TRACK BACK
SVC 171
CALL RETURN AND REQUEST TIME SLOT
DSMGTOUR
DSMPUTDT
DSMPUTDT
FINISH INDEX ENTRY & ADD 1 TO PIECE COUNT
DIAGNOSE: CHECK INDEX TRACK VALIDITY
WRITE INDEX TRACK BACK
SVC 171
CALL RETURN AND REQUEST TIME SLOT
DSMRDTRK

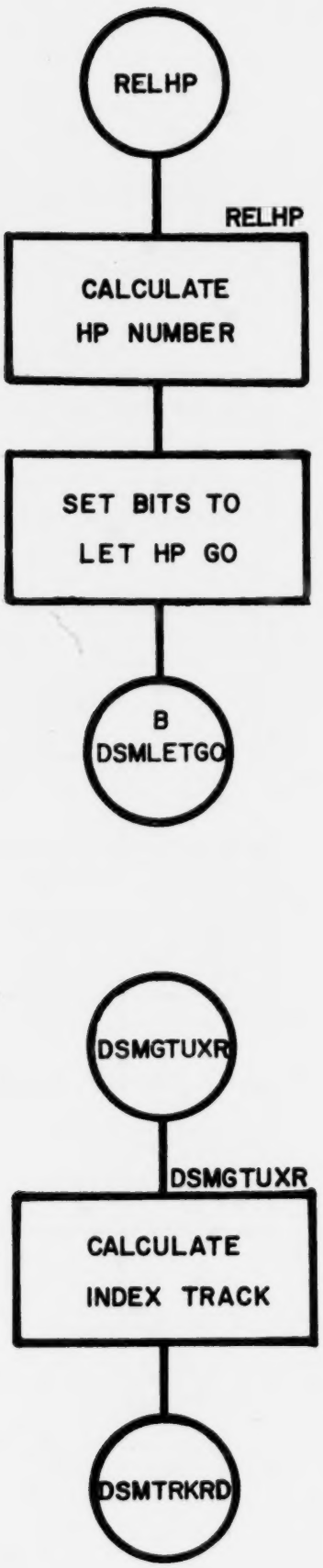

DSMRDTRK
DSMRDTRK
BUILD ADDRESS TO READ DATA TRACK
READ DATA TRACK
SVC 171
CALL RETURN AND REQUEST TIME SLOT
DSMUSEDT
DSMUSEDT
MOVE DATA INTO TRACK
WRITE DATA TRACK
SVC 171
CALL RETURN AND REQUEST TIME SLOT
DSMFEXIT

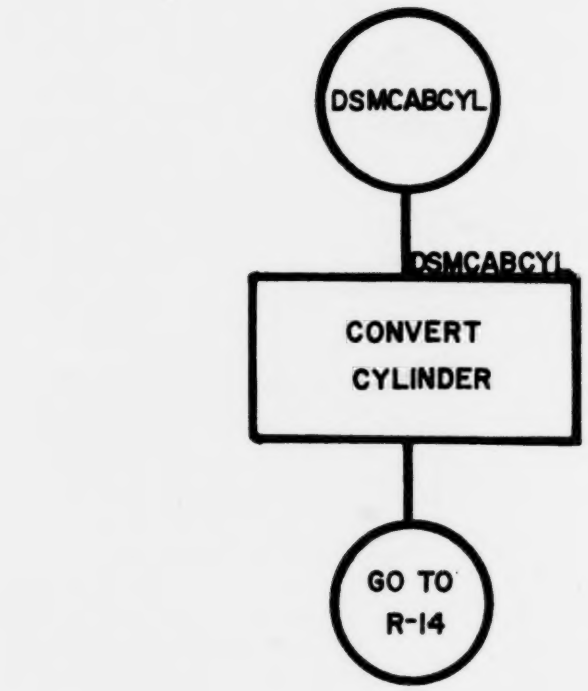

DIAGNOS
DIAGNOS
CHECK ALL COUNTS AND LENGTHS
ANY ERRORS ?
YES
SVCBOMB
NO
GO TO R2
DSMNXT
DSMNXT
IS THERE ANOTHER INDEX TRACK ?
YES
DSMMKCHN
NO
DSMTOK
DSMTOK
READ CURRENT INDEX TRACK
SVC 171
CALL RETURN AND REQUEST TIME SLOT
R3 = INPUT AREA
DSMGTOVF

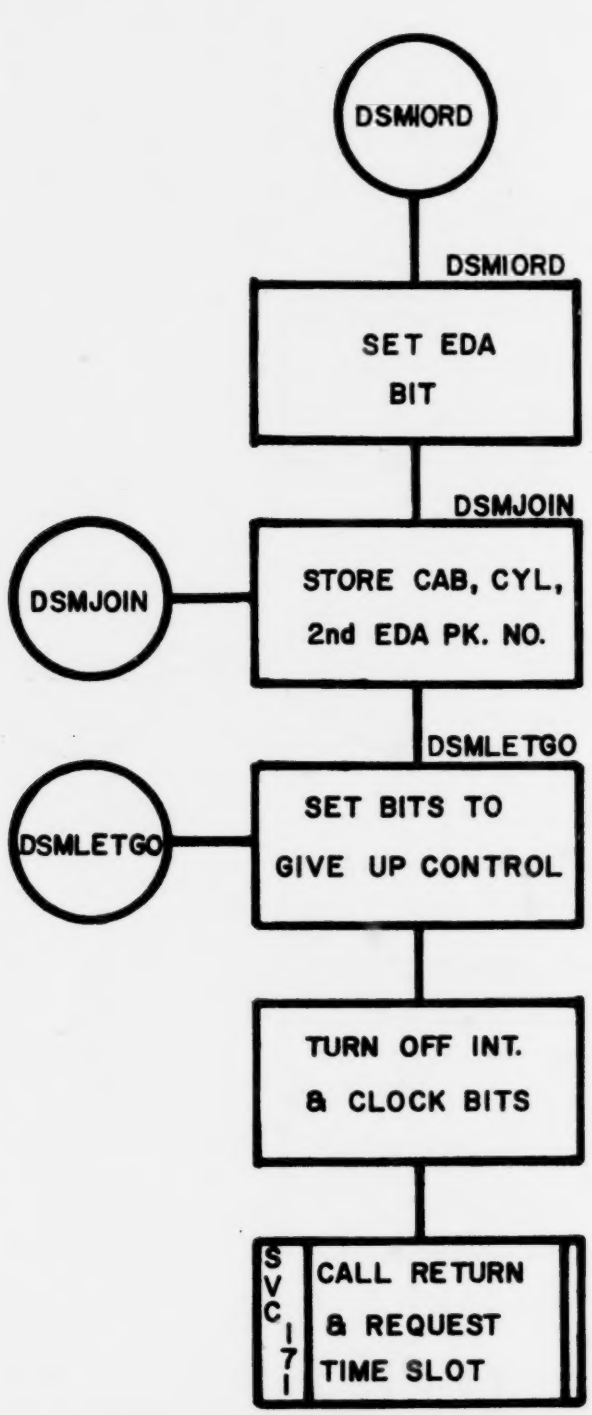

DSMGTOVF
DSMGTOVF
END OF
SUB-MEDIA ?
NO
YES
TO FIRST TRACK
IN SUB-MEDIA
=R2
TO
NEXT TRACK
=R2
DOES
R2 = ORIGINAL
HASH ?
NO
DSMOVRTN
YES
DSMFEXIT

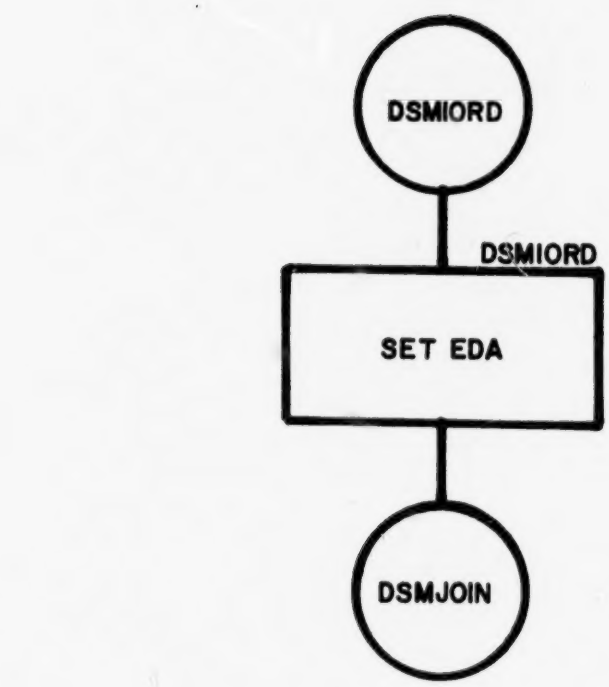
DSMMKCHN
DSMMKCHN
MAKE OVERFLOW POINTER FOR INDEX TRACK
WRITE INDEX TRACK BACK
SVC 171
CALL RETURN AND REQUEST TIME SLOT
ADD ONE TO CURRENT TRACK
DSTKOK

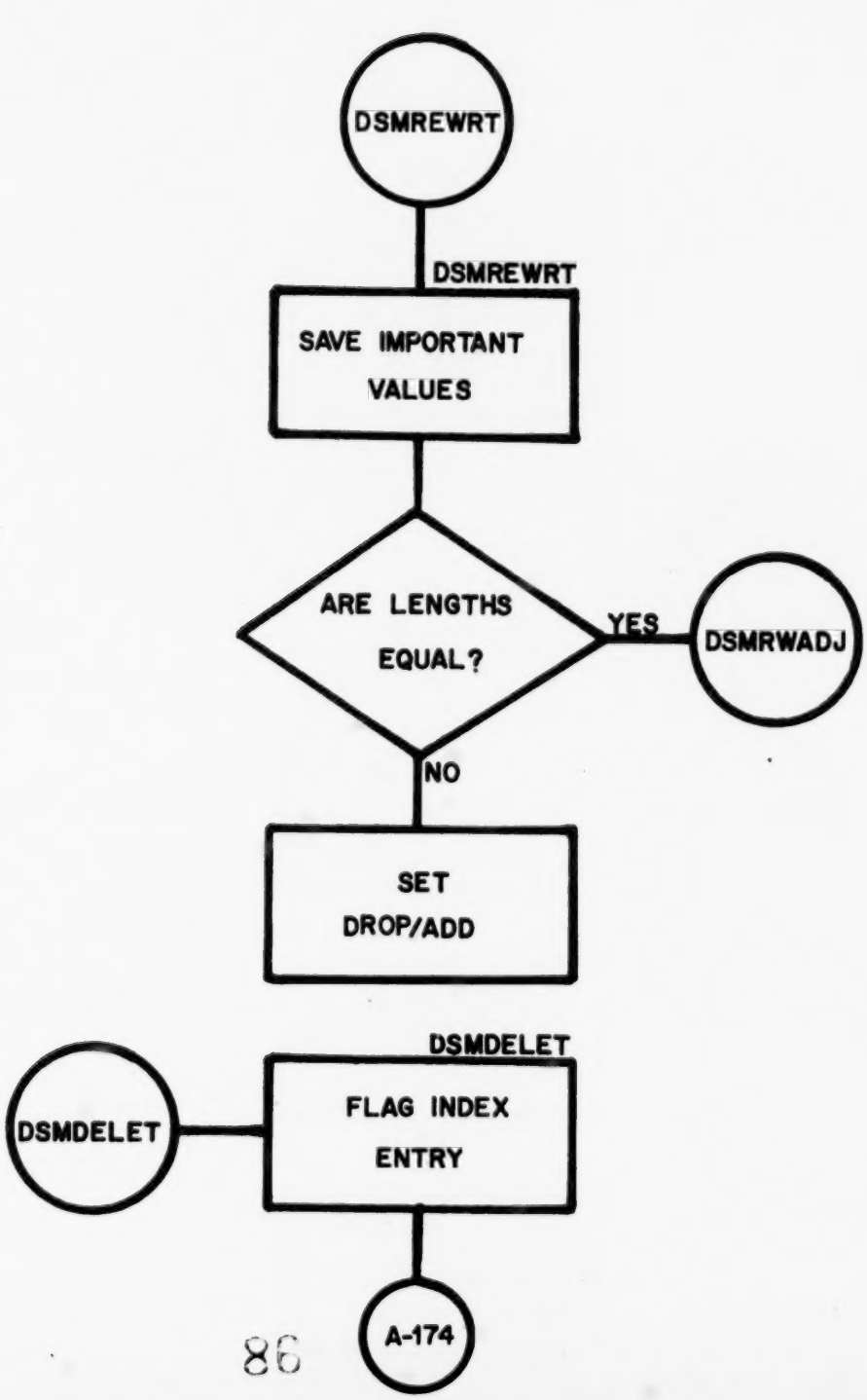

DSMFEX1R
DSMFEX1R
CALCULATE WHERE DATA TRACK WILL BE
READ DATA TRACK
SVC 171
CALL RETURN AND REQUEST TIME SLOT
R3 = START OF RECORD
DSMFEXIT

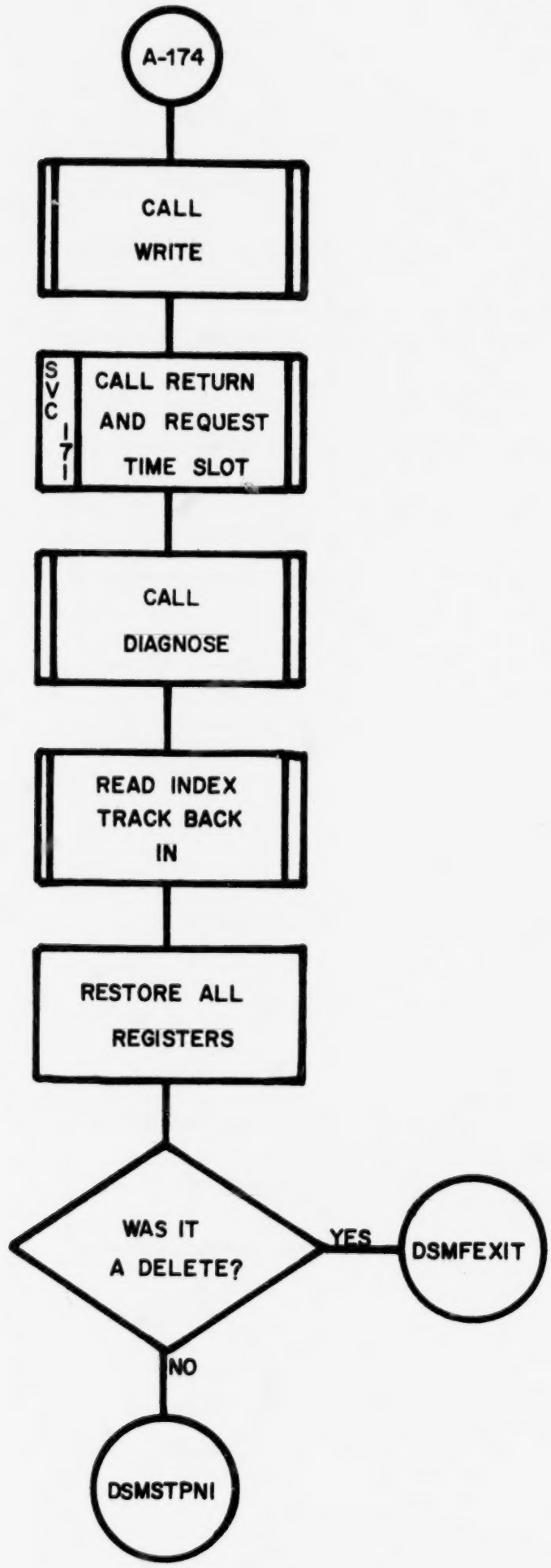

RELHP
RELHP
CALCULATE
HP NUMBER
SET BITS TO
LET HP GO
B
DSMLETGO
DSMGTUXR
DSMGTUXR
CALCULATE
INDEX TRACK
DSMTRKRD

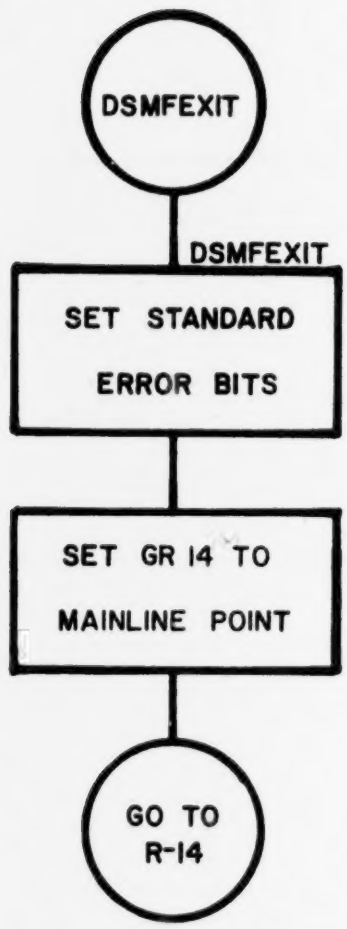

DSMCABCYL
DSMCABCYL
CONVERT
CYLINDER
GO TO
R-14

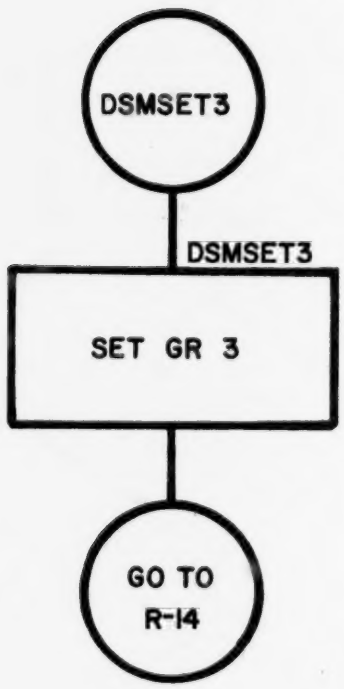

DSMIORD
DSMIORD
SET EDA
BIT
DSMJOIN
DSMJOIN
STORE CAB, CYL,
2nd EDA PK. NO.
DSMLETGO
DSMLETGO
SET BITS TO
GIVE UP CONTROL
TURN OFF INT.
& CLOCK BITS
SVC 171
CALL RETURN
& REQUEST
TIME SLOT

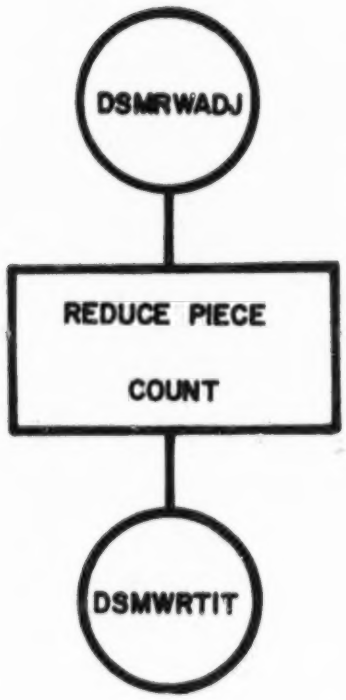

DSMIORD
DSMIORD
SET EDA
DSMJOIN


DSMREWRT
DSMREWRT
SAVE IMPORTANT VALUES
ARE LENGTHS EQUAL?
YES
DSMRWADJ
NO
SET DROP/ADD
DSMDELET
DSMDELET
FLAG INDEX ENTRY
A-174


A-174
CALL
WRITE
SVC 171
CALL RETURN
AND REQUEST
TIME SLOT
CALL
DIAGNOSE
READ INDEX
TRACK BACK
IN
RESTORE ALL
REGISTERS
WAS IT
A DELETE?
YES
DSMFEXIT
NO
DSMSTPNI


DSMFEXIT
DSMFEXIT
SET STANDARD
ERROR BITS
SET GR 14 TO
MAINLINE POINT
GO TO
R-14


DSMSET3
DSMSET3
SET GR 3
GO TO
R-14


DSMRWADJ
REDUCE PIECE
COUNT
DSMWRTIT